Multiple Track Performance of a Digital Magnetic Tape System : Experimental Study and Simulation using Parallel Processing Techniques.

Timothy John Jackson BSc(Hons), AMIEE.

Submitted to the Council for National Academic Awards in Partial Fulfilment for the Degree of Doctor of Philosophy.

Sponsoring Establishment:

Polytechnic South West,
Plymouth, Devon.
School of Electronic, Communication and
Electrical Engineering.

Collaborating Establishment:

Thorn EMI,
Central Research Laboratories,
Hayes, Middlesex.

September 1991.

I dedicate this thesis to

my parents, and Sarah.

Declaration.

I declare that this thesis is the result of my investigation only, and is not submitted in candidature for the award of any other degree. During the research programme I was not registered for the award of any other CNAA, or any other academic institute award.

Acknowledgements.

I would like to express my gratitude to all those people and organisations who have been associated with this project, in particular,

The supervisory team of Professor D.J. Mapps (Director of Studies), Dr. E.C. Ifeachor and Dr. T. Donnelly of the School of Electronics, Communication and Electrical Engineering, for their constructive advice, encouragement and guidance throughout the project.

Dr. A. Vaidya and Mr. P.R. Evans of Thorn EMI Central Research Laboratories for their collaboration in this project.

Those based at The Charles Cross Centre for their support, friendship and times of light relief.

To the Science and Engineering Research Council and Thorn EMI Central Research Laboratories for their financial assistance.

Nomenclature.

x, y, z = The three orthogonal cartesian axis,
x in direction of tape travel
y into the surface of the tape
z across the width of the tape.

ß = Interference function between tracks.

B = Frequency bandwidth, Hz.

C = Shannon Channel Capacity.

d = Head to magnetic medium spacing distance, m.

f_x = Artangent parameter.

f_s = Sampling frequency, Hz.

g = Head gap width, m.

H = Magnetic field, Am^{-1}

k = Wavenumber = $2\pi/\lambda$, m^{-1}

K = Multiplier, 1024.

M = Magnetisation, Am^{-1}

m = Displacement of head, m.

N = Number of tracks.

P = Number of magnetic particles.

r = Read head width, m.

s = Track separation, m.

T = Data bit period or sampling period, s.

V = Velocity of tape, ms^{-1}.

w = Write head width, m.

X = Distance in x direction = Vt, m.

δ = Magnetic medium thickness, m.

Φ = Magnetic flux, T.

λ = Wavelength of recorded information, m.

bps = Bits per second.

GXO = Gated Cross-Over.

LHD = Lateral Head Displacement.

LPS = Linear Pulse Superposition.

LSB = Least significant bit.

MSB = Most significant bit.

PW50 = Isolated pulse width at 50% of it's maximum amplitude.

SNR = Signal to noise ratio.

TTL = Transistor-Transistor Logic.

Abstract.

Multiple Track Performance of a Digital Magnetic Tape System : Experimental Study and Simulation using Parallel Processing Techniques.

by

Timothy John Jackson.

The primary aim of the magnetic recording industry is to increase storage capacities and transfer rates whilst maintaining or reducing costs. In multiple-track tape systems, as recorded track dimensions decrease, higher precision tape transport mechanisms and dedicated coding circuitry are required. This leads to increased manufacturing costs and a loss of flexibility. This thesis reports on the performance of a low precision low-cost multiple-track tape transport system. Software based techniques to study system performance, and to compensate for the mechanical deficiencies of this system were developed using ***occam*** and the ***transputer***.

The inherent parallelism of the multiple-track format was exploited by integrating a ***transputer*** into the recording channel to perform the signal processing tasks. An innovative model of the recording channel, written exclusively in ***occam***, was developed. The effect of parameters, such as data rate, track dimensions and head misregistration on system performance was determined from the detailed error profile produced. This model may be run on a network of ***transputers***, allowing its speed of execution to be scaled to suit the investigation. These features, combined with its modular flexibility makes it a powerful tool that may be applied to other multiple-track systems, such as digital HDTV.

A greater understanding of the effects of mechanical deficiencies on the performance of multiple-track systems was gained from this study. This led to the development of a software based compensation scheme to reduce the effects of Lateral Head Displacement and allow low-cost tape transport mechanisms to be used with narrow, closely spaced tracks, facilitating higher packing densities.

The experimental and simulated investigation of system performance, the development of the model and compensation scheme using parallel processing techniques has led to the publication of a paper and two further publications are expected.

Contents.

1. Introduction.

Since the introduction of the audio compact-cassette in 1963 and the colour video recorder in 1976, magnetic tape has satisfied the storage needs of the domestic market. Pushed by the demand for higher performance, these analogue systems are being superseded by digital systems (for example, the Compact Disc). Digital recording has many advantages: it is highly linear and stable; can be used with very low signal-to-noise ratio channels; and allows degradation-free duplication. Performance limitations of a digital magnetic recorder are primarily governed by the analogue-to-digital conversion process and not the recording process. However it requires a wider bandwidth than analogue recording.

One of the ways of achieving this bandwidth is to use a Rotating-Head recorder (for example R-DAT, the first consumer digital magnetic recorder). The high head-to-tape velocity facilitates a wide bandwidth without requiring a high tape speed. Also, rotating-head recorders can use very closely spaced tracks, efficiently utilising the recording surface of the tape. However, the tape transport mechanism is complex, making it more expensive to manufacture and miniaturization more difficult. The orientation of the tracks precludes splicing, whilst the tape can be subject to wear problems.

Another way of achieving the required bandwidth is to combine several low bandwidth, stationary-head channels. The main benefit of a stationary-head recorder is the simplicity of the tape transport mechanism. The main detraction is the increase in the signal processing requirements (up to N times in an N-track recorder). In the past, these signal processing requirements have been met by the use of dedicated hardware. The potential benefits of increased flexibility and simplified hardware have prompted investigations into the ability of conventional microprocessors to carry out these tasks (Donnelly, 1986 & 1987). Whilst the results confirm the benefits, they also highlight the poor performance of microprocessors in this application. Without sufficient computational performance, either the complexity of the coding algorithms or the maximum data rate will be compromised.

One reason for this poor performance is the disparity between the inherent concurrency of the signal processing and the sequential operation of conventional microprocessors. For example, a four track head may produce four pieces of data in parallel, each requiring several stages of signal processing. This does not create a problem in recorders that use dedicated circuitry, as many circuits may be assembled as there are tasks to be performed in parallel.

A multiple-processor system would appear to be the answer to this computational performance problem. However, until recently such systems have been scarce and expensive. These machines have tended to fall into two camps: supercomputers, e.g. the Intel iPSC (Hockney et al., 1988) designed for 'number-crunching' scientific problems, and therefore not designed to operate under the constraints of a Real-Time system; and array processors, e.g. the ICL DAP (Hockney et al, 1988) normally designed for a very specific task and not for flexibility or ease of programming. (A Real-Time system is one where the correctness of results depend not only on their logical correctness but also on the time they are produced.)

The introduction of the high level language ***occam*** and its processor, the ***transputer***, has changed this situation. Multiprocessor networks with an infinity of sizes and topologies can be assembled from ***transputers***, and programmed in ***occam***. This scalable architecture allows computers to be assembled with almost unlimited potential computational power. They may be programmed in a very straightforward manner, allowing software techniques previously dismissed as computationaly too complex to be realistically considered.

Digital computers are used extensively in magnetic recording research, for example calculating fields, modelling e.t.c. The integration of a multiple-processor computer (that may be programmed as simply as a standard sequential computer) into the digital magnetic recording channel would produce a very powerful research tool. The advantages of being able to perform the research with the same processor architecture and language as the real-time implementation are many.

This application of digital computing to the magnetic

recording channel reverses the established roles of these two technologies. The first digital computer, the ENIAC (Hockney et al., 1988), was built 48 years after the first magnetic recorder (Smith, 1888), and soon digital magnetic recording was being used to provide a cost-effective, reliable, non-volatile method of storing computer programmes and data.

Magnetic recording continues to meet the demands of the computer industry, indeed it can be argued that the recent explosion in use and popularity of the digital computer would not have come about if it were not for the advances that have been made in digital magnetic recording. Although disc based magnetic storage systems have dominated the computer industry (primarily because of their superior access times compared to tape systems), magnetic tape is still widely used for 'back-up' or long term storage due to its low-cost per bit, and ability to store large amounts of data.

1.1. The Magnetic Recording Process.

The basic elements of the longitudinal magnetic tape recorder channel are shown in figure 1.1. The magnetic tape is normally a very thin ribbon of plastic (often Polyester) that has had a magnetic material (often powdered gamma-Ferric Oxide) bonded to one surface. The information to be recorded is first encoded and conditioned for the recording channel, and then amplified to drive current through the winding of the record head. This current produces a magnetic field, a portion of which bows or fringes out around the gap. The flux from this fringing field links with the magnetic tape coating, magnetising the region of tape directly beneath the gap, with the final imprint determined by the field beneath the 'trailing edge' of the gap. As the tape coating will have been chosen to be magnetically 'hard', it remains magnetised after it has passed the field. The magnitude and direction of the magnetised region will be proportional to that of the recording current.

The primary method of retrieving the longitudinally recorded information is to use another (or the same) inductive type head.

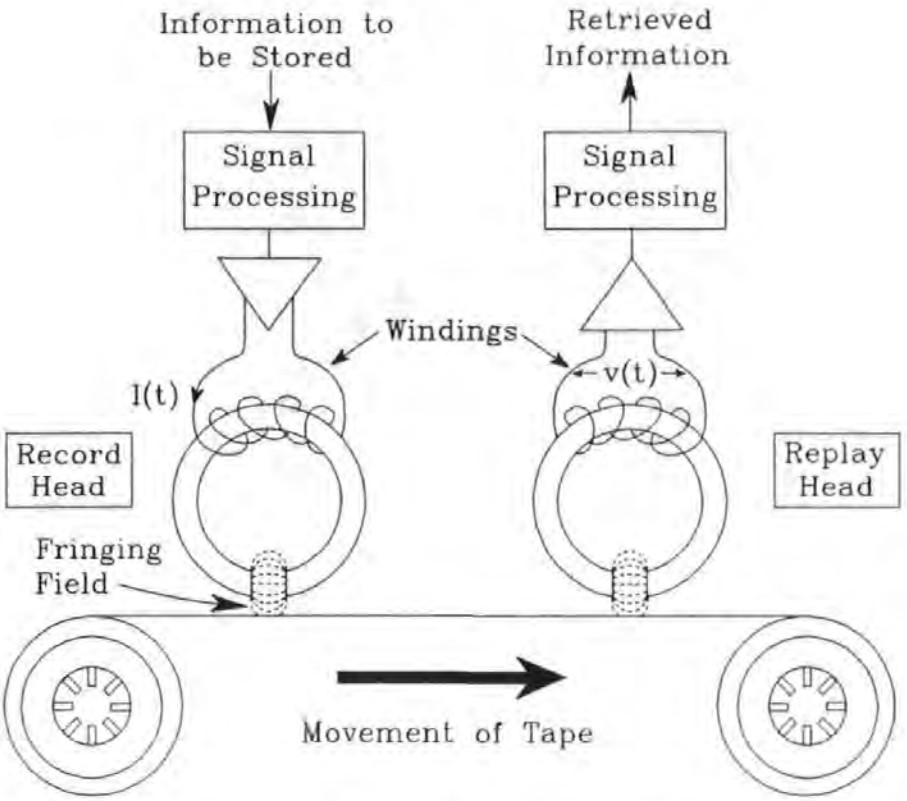


Fig. 1.1. The Magnetic Tape Recorder Channel.

Recent developments in the field of thin films have resulted in the production of a non-inductive read element (being magneto-resistive in operation). Unspecified references to recording heads imply inductive heads (section 2.4.4 covers magneto-resistive heads).

On replay, the tape is moved back across the head. The flux from the magnetised regions of tape links with the magnetic circuit of the head. From Faraday's Law, a voltage is induced in the head's winding proportional to the rate of change of flux, and therefore proportional (albeit, non-linearly) to the original signal. The reproduced voltage may be calculated theoretically from the various magnetic recording parameters and the magnetic and magnetisation fields. Although mathematically complex, a number of simplifications may be made that reduce the complexity to reveal a number of simple relationships. The detail of this is given in Appendix B.

This simple model of the recording process stated above does

not include any non-ideal characteristics. The three main non-ideal characteristics are:

i) Noise, introduced at the record and/or replay stage. This includes thermal (or Johnson) noise and Shot noise (from the signal conditioning electronics) Barkhausen noise in the head and particulate noise from the magnetic medium.

ii) The magnetic recording process itself is non-linear. The signal induced in the replay head winding is non-linearly related to the original record current.

iii) The magnetic recording process is frequency bandlimited. For an inductive system, the upper and lower frequency response limits are determined primarily by the geometry and dimensions of the record and replay heads.

There are many other non-ideal characteristics, but these effects can be reduced to a greater or lesser extent depending on the signal conditioning and encoding scheme used. There are three primary modes of signal conditioning and encoding: Direct recording of analogue signals; Frequency Modulation (FM) of analogue signals; and Digital recording. This project is solely concerned with the last of these three.

1.1.1. Digital Magnetic Recording.

The word 'Digital' in the above section heading refers to the recording process and not the information content of the signal to be recorded. Analogue information may be stored using a digital magnetic recorder after the appropriate analogue to digital conversion. This text is concerned with digital magnetic recording, and does not address the conversion to and from the digital domain.

Digital magnetic recording has many advantages. The system's accuracy, linearity and dynamic range are not limited by the recording process, but by the pre-recording signal processing and

coding. Also, it is simple to multiplex several digital signals down a single recording channel. A key factor, that will be exploited extensively in this project, is the ease with which digital recording systems can be integrated with computer systems, and the opportunity this gives to exploit powerful digital signal processing techniques.

The simplest form of digital recording is NRZ-L (Non-Return to Zero, Level). It employs virtually no conditioning or encoding. The two binary voltage levels are directly converted into two current directions that saturate the magnetic medium North or South (with respect to the tapes direction).

During replay, a quantitative knowledge of the magnetic field strength, as is required for direct recording of analogue signals, is not required. It is only necessary that adjacent and opposing magnetic regions are of sufficient magnitude for the transition between then to be detectable. It is this latitude (between transition and no transition) that results in digital magnetic recording being far less susceptible to two of the three non-ideal characteristics listed previously; noise and non-linearity. Unfortunately the remaining non-ideal characteristic, that of limited bandwidth, is compounded, as digital magnetic recording requires an even greater bandwidth. Modern encoding schemes can be designed to reduce this problem.

A solution to the limited bandwidth (and also to the desire for higher bit packing densities) can be seen from an extension of Shannon's Channel Capacity (Mallinson, 1987(a)), where the capacity, C, of the channel is defined by,

$$C = B.\log_2(1 + SNR_W) \qquad \text{Equ. 1.1}$$

where B = Bandwidth of the Channel

SNR_W = Full Tape Width Signal-to-Noise Ratio

The SNR may be approximated by (Mallinson, 1987(b)),

$$SNR_W = \text{Full Tape Width Signal-to-Noise Ratio} = \frac{wp\lambda_m^2}{2\pi} \qquad \text{Equ. 1.2}$$

where p = Number of magnetic particles per unit volume

w = Track Width

λ_m = Minimum Recorded Wavelength.

Therefore, if the width of the tape is divided into N tracks, then (ignoring inter-track guard-bands) the SNR of each track is reduced by 1/N, giving a total channel capacity of,

$$C_N = N.B.\log_2(1 + \frac{SNR_W}{N}) \qquad \text{Equ. 1.3}$$

Figure 1.2 shows how the channel capacity varies with the number of tracks, assuming a SNR of 50dB and bandwidth of 12kHz (approximate values for the compact-cassette system). The capacity can be seen to increase nearly linearly with the number of tracks (as will the areal packing density, assuming the same head-to-tape velocity). In addition, equation 1.2, shows that halving the track width reduces the SNR by 3dB, whilst halving the minimum recorded frequency reduces the SNR by 6dB. Therefore, although doubling the recording frequency may seen to be the simplest solution for the designer, there are important performance advantages to be gained from doubling the track density instead.

Rotating-head recorders use a slow tape speed and a high rotating-head drum speed for closely spaced tracks, high bandwidth and packing density. In a stationary-head multiple-track recorder, the track widths and inter-track guard-bands are reduced to fit as many tracks across the width of the tape as possible. However, increasing the packing density in these ways increases the following problems.

i) As tracks get narrower, the SNR becomes worse (a loss of 3dB for each halving of the track width).

ii) Drop-outs affect larger amounts of data.

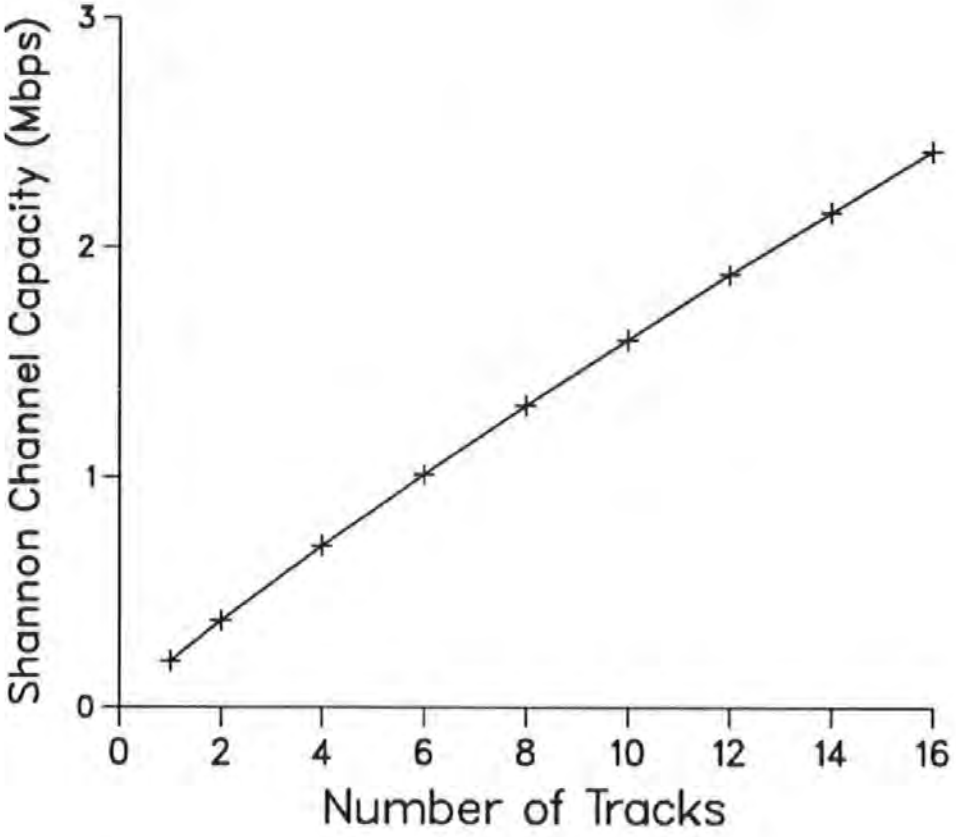


Fig. 1.2. Relationship between Channel Capacity and Number of Tracks (SNR_W=50dB, B=12kHz).

iii) Crosstalk increases as tracks and their respective head assemblies get physically closer.

iv) Mechanical deficiencies in the tape transport (like tape skew and head alignment) become more pronounced.

Addressing these points in order:

i) The realisable channel capacity of magnetic recorders is a fraction of the Shannon Capacity. For example, a typical state of the art recorder may have a SNR of 30dB and bandwidth of 2MHz, yet only achieve a data rate of 4Mbps (Mallinson, 1987(b), p125). The Shannon Capacity of such a machine is 10Mbps. A recorder with the same bandwidth capable of operating at the Shannon Capacity would require a SNR of just 3·0dB to achieve the same data rate. Therefore the

reduction in SNR resulting from narrower tracks may be considerably offset by using more sophisticated coding schemes that allow operation nearer the Shannon Capacity.

ii) The effects of drop-outs may be dealt with by a suitable error detection and correction scheme.

iii) To date the philosophy has largely been to reduce the mechanical deficiencies at source rather than cope with the problems they cause (this has previously proved fruitful). For example, the problem of tape-to-head misregistration has been dealt with by tightening the manufactured tolerances of the tape, tape guides, bearings etc. This obviously increases manufacturing costs. As the 'cost of computation' is falling it is desirable to develop software techniques to tackle these problems.

It is important to note that the tape is included in the list of components that need to be manufactured to tighter tolerances (for example, at the tape slitting process). This increases the cost of every tape. A software solution to these problems would result in a one-off increase in cost, and this, with the advances being made in microprocessor technology, may well become insignificant.

iv) The solution to the problems caused by track misregistration may well solve those caused by cross-talk, as they both distort and corrupt signals in a similar manner. Electronic compensation for non-varying cross-talk already exist (e.g. van Gestel et al., 1982), and may be transferred to software.

Therefore the last of these problems to be solved or offset by software techniques are those caused by mechanical deficiencies. This project was particularly interested in the problems caused by lateral head displacement.

1.1.2. Channel Coding

Channel codes are used to match the characteristics of the data stream to that of the recording channel. For example, figure 1.3 shows the very poor response of the replay head at low frequencies, producing no signal at 0Hz. Frequency equalisation of the replayed signal may be used to compensate for the bandlimited nature of the magnetic recording response. Equalisation however, will not help solve the DC response problem when used with a code like NRZ-M code (Non-Return-to-Zero, Mark). NRZ-M records a flux transition for a datum '1' and no transition for a datum '0'. A continuous stream of '0's will result in no flux transitions; effectively a 0Hz signal that the head will not respond to. This obviously makes synchronisation and retiming of the waveform very difficult. Removing or reducing the low frequency content of the data stream is therefore often a prime objective of a channel code.

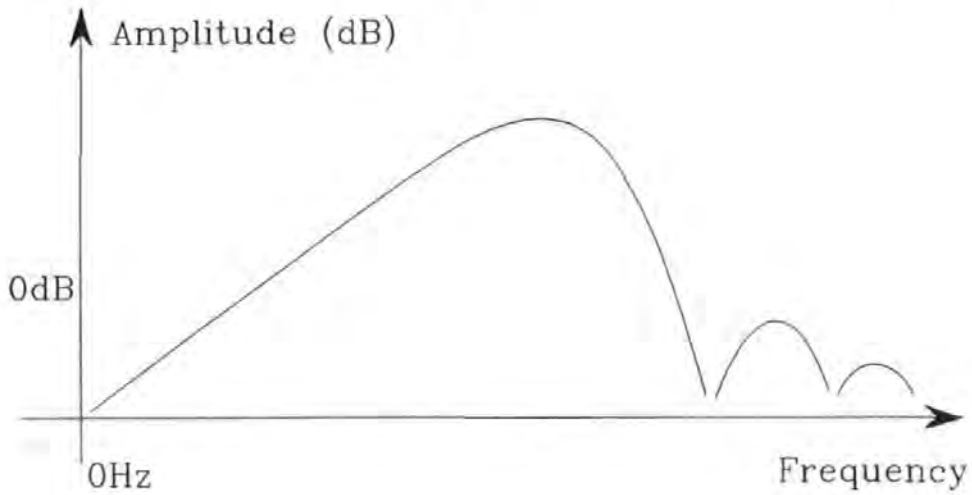


Fig. 1.3. Frequency Response of a typical Replay Head.

The Bi-Phase-L channel code completely removes the DC content of the signal by mapping each data bit into a two bit code word (which is then recorded using NRZ-M):

Data Bits	Code Bits
0	1 0
1	0 1

As each code word is DC free, any sequence of code words will also be DC free. There is a transition at the centre of each code word, and therefore regenerating the clock signal for synchronisation and retiming is straightforward. As Bi-Phase-L records (on average) more transitions than NRZ-M for the same data sequence, a wider bandwidth channel is required or the data rate must be reduced. The choice between ease of clock regeneration and bandwidth is typical of the type of compromise a channel code will introduce.

Three parameters (ignoring charge constraint) fully describe the characteristics of the many channel codes developed:

d = minimum distance between transitions in coded sequence.

k = maximum " " " " " "

R = Rate of the Code = (m/n) where,

m = number of bits of data to be coded.

n = " " " in coded sequence.

From these three parameters, the following three primary characteristics may be calculated:

Detection Window = R * data bit cell period = R * T

(a wide Detection Window eases sampling)

Max:Min Pulse Width Ratio = (k + 1)/(d + 1)

(low values are desirable as this suggests small peak-shift and good self-clocking, amongst others)

Density Ratio = R * (d + 1)

(high values suggest high signal-to-noise ratio)

Table 1.1. summarises some of the more popular channel codes, and illustrates how the various characteristics have been compromised. More detail on this subject can be found in Jorgensen's book (Jorgensen, 1988), from which the data for Table 1.1. was based, and the paper by Mackintosh (Mackintosh, 1979(b), that includes clear descriptions of the established channel codes. However, for a

thorough treatment of the subject that concentrates on the design of more recent codes (especially Run Length Limited and DC-free codes), Schouhamer-Immink's book (Schouhamer-Immink, 1991) is suggested.

	d	k	m	n	Code Rate	Detection Window	Pulse Width Ratio	Density Ratio
NRZ-L	0	∞	1	1	1	T	∞	1
NRZ-M	0	∞	1	1	1	T	∞	1
Bi-Phase-L	0	1	1	2	$1/2$	$T/2$	2	$1/2$
MFM	1	3	1	2	$1/2$	$T/2$	2	1
E-NRZ	0	7	7	8	$7/8$	$7T/8$	8	$7/8$
Miller^2	1	5	1	2	$1/2$	$T/2$	3	1
ZM	1	3	1	2	$1/2$	$T/2$	2	1
3PM	2	11	3	6	$1/2$	$T/2$	4	$3/2$

Table 1.1. A Comparison of Various Channel Code Parameters.

1.1.3. Error Correction Coding

Digital magnetic recorders that use saturation recording (that is the norm) benefit from a broad gap between detected states. Although this can result in a high degree of immunity to noise, errors will still occur. The principal cause of errors in an optimally set-up system is tape drop-out. This is a (normally) short and severe loss of output from the playback head, caused by: tape surface imperfections; physical damage; or debris between the tape surface and head.

Error coding involves adding extra information to the data, allowing errors in the data at replay to be detected (and usually corrected). An intuitive example would involve recording the data three times, and using a majority voting logic circuit to decide the most probable result.

There are two main types of error correcting code in use, (i) block codes and (ii) convolution codes. Note, the standard nomenclature confusingly duplicates symbols used in the characterisation of channel codes.

i) A block code splits the data stream to be encoded into message blocks of length k bits. Each message block is converted into a code word of length n (n greater than k). This is termed an (n,k) block code. Block channel codes are memoryless, and therefore each code word is dependent only on the k bits of the current message block.

Therefore of the 2^n possible code words, only 2^k are used. If a word is received that is not one of the allowable 2^k code words then an error has occurred. In many systems the code word that should have been received is determined by calculating which of the valid code words is nearest to the one received. The reliability of this type of code is largely dependent on the probability of the corrupted code word not being a valid one.

(ii) A convolution code also uses k length sequences of data and produces n length code words. However, the code words are dependent not only on the k bits of this message, but on m previous message bits as well, and is therefore called an (n,k,m) convolution code.

Error correction algorithms vary from simple correction schemes, like Hamming, which have a correspondingly modest error correction capability, to complex schemes, like Reed-Solomon, that have very powerful error correction capabilities. For a thorough introduction to the subject, Hill's book (Hill, 1986) is recommended. Blahut's book (Blahut, 1983) covers more complex schemes (like BCH codes and Spectral techniques) with much emphasis on their

implementation.

1.2. The Compact-Cassette Tape Format.

Philips introduced the compact-cassette at the 1963 Berlin Radio Exhibition. It was designed to record mono-aural audio, primarily for the domestic market. Recordings are made on one half of the tape, then the cassette is manually turned over to record onto the other half. The track format was extended in 1966 to allow stereo recording, see figure 1.4. The compact-cassette gained popularity as a recording medium for music due to its low cost, robustness and convenient size.

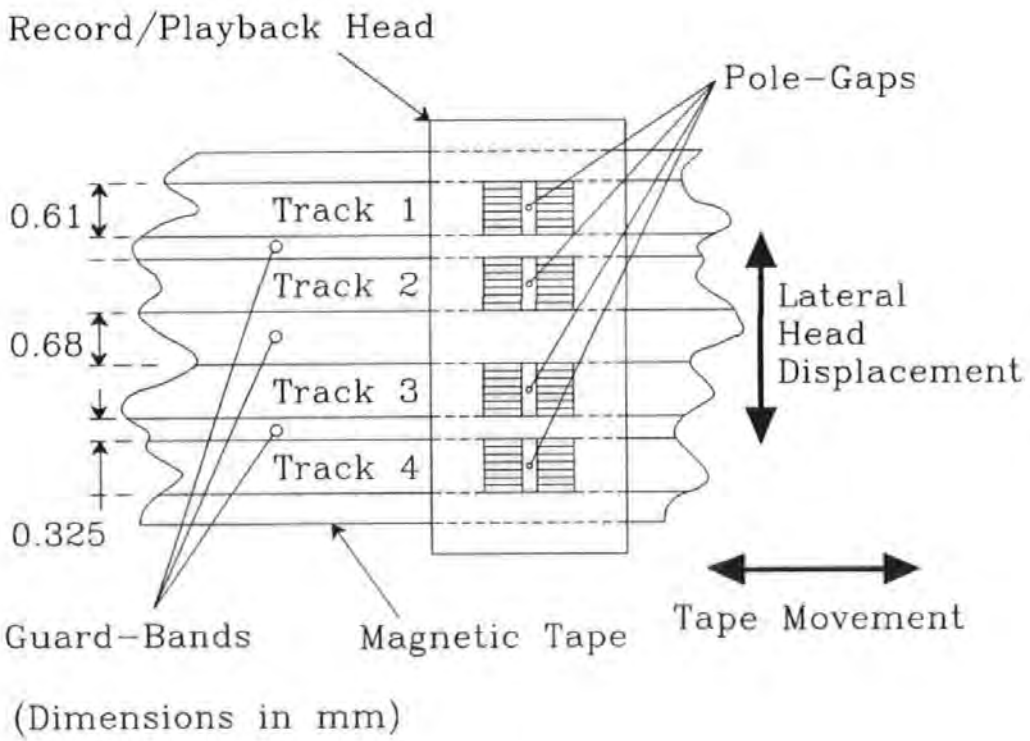


Fig. 1.4. The Compact-Cassette Track Format.

The choice of the compact-cassette as the recording format for a digital recorder project may thus seem unusual, especially as many other tape formats have been developed since the compact-cassette, some specifically developed for digital recording. The reasons for

its choice were:

i) It is a stationary head design. Rotating head designs are very popular at present, but the manufacturing cost of a rotary head mechanism is much higher than a stationary head design because it involves more moving parts, manufactured to tighter tolerances.

ii) The mechanisms are simple, and allow modifications to be made easily (see section 2.4.2).

iii) It is a multiple-track format. One aspect of the project was to investigate software processing of data from a multiple-track format.

iv) The simplicity of its design should make techniques developed using it be applicable to other systems.

v) It is cheap to buy and widely available as an established system.

One of the aims of the project was to keep the number and complexity of mechanical components to a minimum, (thereby reducing cost), and to investigate how software techniques may be used to compensate for those deficiencies that result, actual and envisaged. (It is expected that the cost of computational performance will continue to drop faster than mechanical manufacturing costs).

Several commercial digital-audio systems based on the compact-cassette have been developed. From research papers published, at least two companies appeared to be about to go into production of such systems around 1984: Matsushita Electric Company (Sakamoto et al., 1984) and Mitsubishi Electric Company (Onoshi et al., 1984). For reasons unknown, no such systems have been commercially released. Philips are about to to change this situation.

1.2.1. The Philips DCC Format.

Philips are due to release a new digital magnetic tape format called Digital Compact Cassette (DCC) (Cole, 1991 and Fox, 1991). The DCC cassette's housing is physically very similar to the analogue

compact-cassette, but eighteen tracks are recorded across the tape, nine at a time (eight dedicated to audio). As with the analogue format, the head records across half the tape width at a time, but it is the head (and not the cassette) that is automatically turned over in a DCC mechanism to access the other half.

Although each of the eight tracks of the new DCC recording head can record shorter wavelengths signals than the compact-cassette head (1μm compared to 2.5μm), the system cannot process data at the rate of $2x10^6$ bits per second as can the Compact Disc (CD) (Watkinson, 1989). To provide a comparable level of 'audio quality' as CD, the bit rate is reduced using a coding scheme called PASC (Precision Adaptive Sub-band Coding). PASC coding splits the signal into 32 frequency bands, and then (using a template of the human ear's frequency sensitivity) encodes only those sounds that are above the threshold of audibility. By not coding 'inaudible' sounds the number of bits per sample is reduced from 16 to an average of 4.

The design of the recording head appears to be similar to the one proposed in section 2.4.4, but there is as yet little definitive information. The head is split into two sections: an inductive Ferrite section for writing; and a Magneto-Resistive section for reading. The track format is shown in figure 1.5. To reduce the effects of lateral tape-to-head mis-registration, the read element is 70μm wide compared to the write track width of 185μm.

1.3. Data Processing.

1.3.1. Hardware or Software Processing ?

Conventionally, the signal processing and coding involved in the recovery of data from digital magnetic recorders has been carried out by dedicated hardware. Figure 1.6(a) shows the signal processing tasks of a typical replay channel. As the project was concerned with using software techniques, the question that was addressed was how much of this channel should be implemented in hardware and how much in software ?

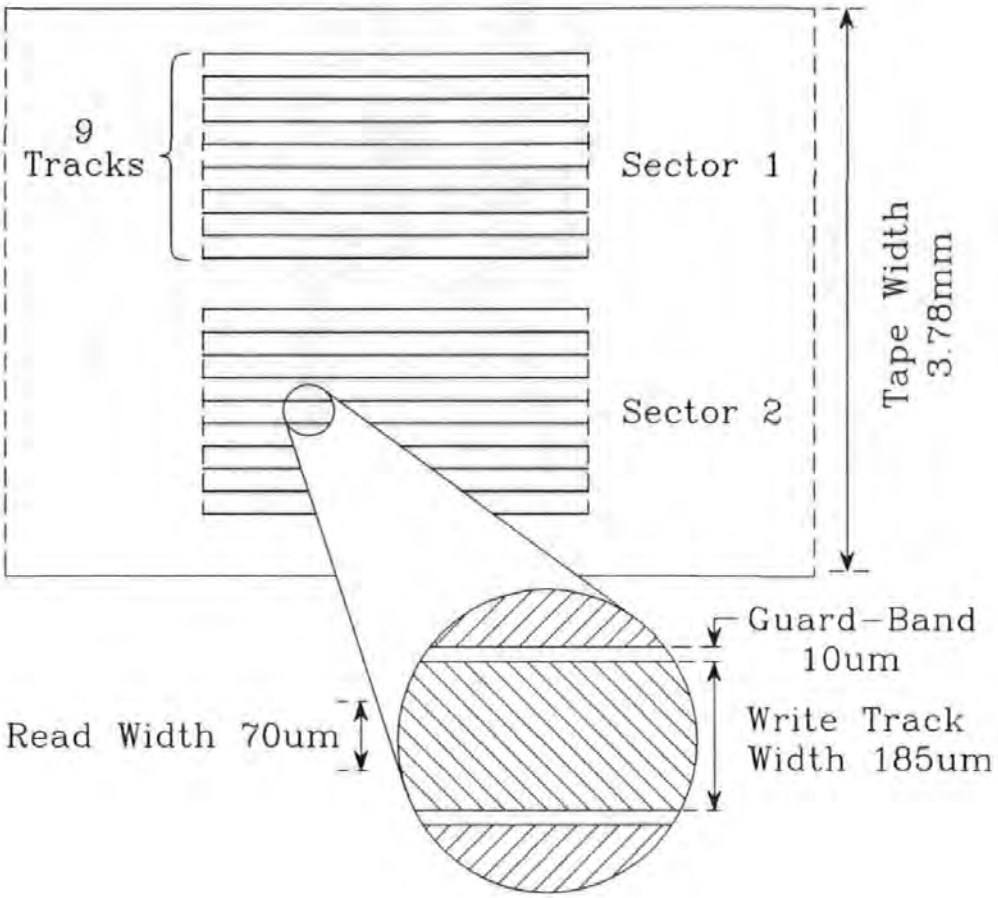


Fig. 1.5. The DCC Track Format.

Figure 1.6(b) shows what might be called the 'ultimate' software solution. Hardware is only used to amplify and digitise the signal prior to input to the computer. This solution would impose almost no limitations on the type or complexity of signal processing implemented.

In a real-time system, the complexity of the signal processing of each bit is limited to what can be done in a bit period. There is therefore a compromise between algorithmic complexity and maximum data rate. The more tasks that are performed in hardware, the simpler (and therefore the faster) the software, and consequently the higher the maximum data rate. As the information content of the signal reduces passing through the channel, the earlier in the signal processing chain the software takes over, the more information

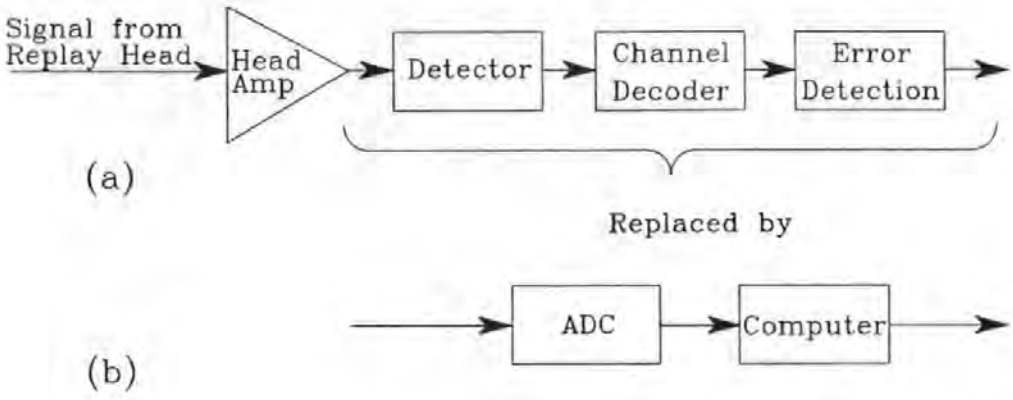


Fig. 1.6. Signal Processing of the Replay Channel.

it has to work with, and therefore the greater the potential to recover the original data.

A single, general purpose microprocessor will be very limited in the complexity of algorithm it can implement and/or data rate it can sustain. Its sequential architecture is not suited to the parallel nature of the tasks involved in the recovery of data from a multiple-track digital magnetic recorder. Whilst processors specifically designed for signal processing tasks would be better suited to the evaluation of many of these tasks (Sandler et al, 1989), they are lacking in terms of support for parallel systems.

1.3.2. Concurrent and Parallel Processing

To avoid ambiguity, the following definitions (Galland, 1982) have been adhered to:

- "Concurrency: The condition of multiprogramming; . . ."
- "Multiprogramming: . . . in which two or more application programmes are being executed simultaneously by the interleaved allocation of a single set of computer resources."
- "Parallel Processing: . . . executed at the same time using multiple sets of facilities."

Concurrency can therefore be thought of as pseudo-parallelism. Historically, any concurrency in a system was at the operating system level. For example, the UNIX operating system shares the computers resources (often based around a MC680x0 processor) between several users, each of which may be running several application programmes. This sharing task is handled by UNIX's Kernel, a piece of software. The success of UNIX demonstrates that the benefits of a multi-user, multi-tasking environment can outweigh the performance degradation introduced by the Kernel.

Parallel processing arose primarily from the desire for increased performance. Their specific languages developed from the need to programme parallel processing machines. For example, the CRAY-2 supercomputer (Hockney et al., 1988) has four main processing units, each with its own memory systems and multiple arithmetic units. It has a peak theoretical performance of $2x10^9$ floating point operations per second. Several standard languages (e.g., C and Fortran) are available, their compilers producing code that makes optimal use of the multiple processing elements.

These languages were developed for sequential processing architectures and have been extended to include support for multiple processor architectures. The next logical evolutionary step was to develop languages with support for parallelism designed-in from inception.

1.3.3. The *occam* Programming Language.

Occam (INMOS 1988(a)) is a high-level language designed for parallel programming. ***Occam*** programmes are constructed from Processes. An ***occam*** Process is a named group of instructions that perform a specific task (similar to a Procedure in a standard sequential language). The key point is that ***occam*** allows these processes to be constructed to run either in parallel or sequentially with virtually equal ease. The language supports, but makes no distinction between, parallelism and concurrency. Therefore an ***occam*** programme is not specific to any particular topology or

number of processors. This not only allows great flexibility, it allows programme development initially to be carried out on a single processor, and then transferred to a multiple processor network if available.

The decision between executing a set of processes concurrently or in parallel is made after the programme has been compiled: at the Configuration stage (INMOS 1988(b)). Configuring an ***occam*** programme allocates processes to processors. If there is only one processor then it will be allocated all processes for concurrent processing (that is interleaved sharing of resources). If there is more than one processor, the processes are divided between them for parallel processing.

Many events in the 'real-world' occur simultaneously or overlap: they are not limited to a sequential ordering. A parallel algorithm must first be transformed into a sequential form if it is to be coded in a sequential language. This transformation will tend to hide or mask the relationship between the solution and the implementation of the solution, as well as introducing another step into the coding operation, increasing the probability of a coding error. There are therefore obvious advantages to be gained from using a computer language that allows a similar construction of events.

The speed of execution of a programme written in a parallel processing language is primarily dependent on the number of processors the code is distributed over. In practice, the ultimate performance will be governed by the granularity of the algorithm. In other words, there will be a finite number of tasks that can take place in parallel, and once there is a processor dedicated to each of these, performance increases cannot be gained by using more.

1.3.4. The INMOS *transputer*.

Standard microprocessors provide no support for the parallel programming aspects of ***occam***, like concurrency and inter-processor communication. As standard microprocessors were not designed to be connected into multiple processor arrays, it is not

surprising that they do not do so readily or efficiently. The INMOS Family of *transputers* (INMOS 1988(c)) was designed to satisfy this need.

Transputers are self contained microcomputers on a single integrated circuit, designed primarily to execute *occam* processes. A single *transputer* can perform concurrent processing, whilst a network (of more than one) can perform parallel processing. More detail of the *transputer* can be found in section 2.2.1.

1.4. Previous Work.

In Donnelly's Ph.D. thesis (Donnelly, 1989) several digital magnetic recording systems were developed. All were based on the compact-cassette tape format, and used a conventional microprocessor (a Zilog Z80) for the data processing. Different recording heads, channel codes and error correction strategies were investigated. Several points pertinent to this project were raised:

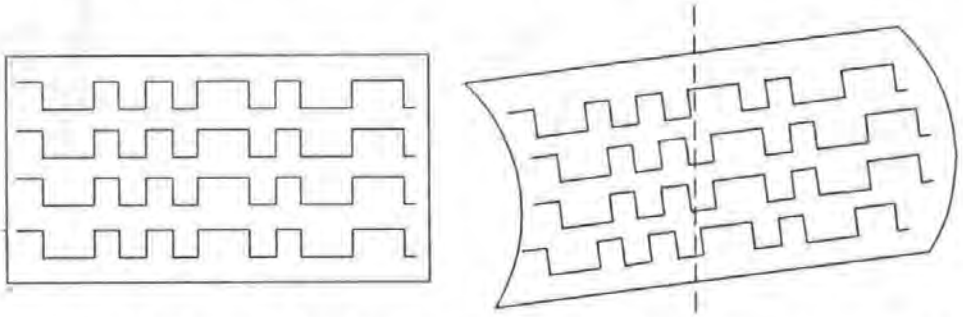
i) Several limitations of the compact-cassette mechanism were offset using software based techniques. In particular tape velocity variations and head azimuth variations.

ii) Tape-to-head mis-registration would assume more significance in a narrower track system.

iii) Data were recorded across all tracks simultaneously, i.e., a four bit word was recorded across the width of the tape in the four track systems, see figure 1.7(a). During playback all tracks were sampled in parallel. This parallel sampling technique reduces the sampling rate by 1/N for an N track system compared to sampling each track individually.

Due primarily to tape skew and tape deformation, the individual bits of the recorded word are skewed with respect to one another on replay, see figure 1.7(b). If the data skew between any two tracks is less than the sampling window width, the recorded word may still be correctly sampled. Sampling window width decreases as

the data rate increases. Data skew in excess of the code bit period at 5kbps was measured. This misalignment of data at the sample point was a significant source of errors. The effective sample time window was doubled using software techniques, reducing the effect of the phenomenon. Donnelly suggested that treating each track independently would remove the problem of mis-aligned data, but would place a further heavy computational burden on the software.



4 Tracks of Data Written in Parallel across width of Tape

On Replay Data becomes Skewed.

(a) (b)

Fig. 1.7. Parallel Recording and Replay.

iv) The error detection method employed in the second recording system (Donnelly, 1989, section 4.4) does not reveal which of the four tracks was the cause of the error, or whether more than one tracks was in error.

v) Reed-Solomon codes were investigated and finite field arithmetic algorithms implemented, but not in real-time due to the amount of computation required.

Simulation may be used to investigate many aspects of the digital recording channel. Mackintosh carried out such an investigation using superposition of isolated pulses to generate the replayed waveform (Mackintosh, 1979(b)). In such an investigation it is important to use a pulse shape that is representative of the pulse produced by the system to be simulated. As Mackintosh was primarily interested in coding schemes, a 'general purpose' shape pulse was

chosen. Mackintosh concluded that a good general 'fit' could be obtained using the equation:

$$f(x) = \frac{1}{(1+x^2+x^4)} \qquad \text{Equ. 1.4}$$

but the often used Lorentzian pulse:

$$f(x) = \frac{1}{(1+x^2)} \qquad \text{Equ. 1.5}$$

was a poor representation. Neither of these basic pulse shapes (nor any of the others Mackintosh investigated) proved to be a good representation of the pulse produced by the compact-cassette system, and so (as reported in section 3.3.2.2) a new basic pulse was derived.

Mackintosh did not directly introduce noise or errors into his simulation work. Instead their effect on the position of transitions was simulated. Transitions were moved from their correct position by a percentage of the PW50 (the width of the isolated pulse at 50% of its maximum height), referred to as RTE (Real-Time Error). Four values of RTE were used, ranging from 0% (indicating no noise) to 16% (8% representing a typical system). All error and noise sources were therefore 'lumped' together. The model described in section 3.3 treated error sources separately as far as was possible, the magnitude of each being based on measured or calculated values.

It is of interest to note that a CDC7600 parallel processing supercomputer was used to run the simulation programme. Twenty years later, a single ***transputer*** was used in a similar role.

1.5. Summary.

In the quest for higher areal bit densities, increasing the number of tracks per mm (by reducing their width and separation) has a fundamental advantage in terms of SNR over increasing the number of

flux reversals per mm. But an advantage of increasing the flux reversal rate is that the data transfer rate may also be increased. One way of compensating for this is to use a parallel track format.

Parallel track systems require more signal processing: up to N times more for an N track system. This naturally suggests a parallel processing architecture, be it at the hardware or software level. Parallel track formats also have their own characteristics that can be exploited advantageously, such as the use of Adaptive Cross Parity (Patel, 1985).

As track widths get narrower, manufacturing tolerances of the tape transport bearings and guides need to be tightened to keep mechanical deficiences, such as head-to-track misregistration, at an acceptable level, leading to increased costs. In contrast, the cost of computation is falling, and is set to carry on doing so. It therefore relevant to investigate how software techniques may be used to address the problems introduced by mechanical deficiences.

The flexibility and potential capabilities of software are not in question. Some of the benefits of using software in the data channel have already been demonstrated (Donnelly, 1989). But digital magnetic recorders are real-time systems, and computational performance is a critical factor. Each data bit has only a limited amount of time to be processed in, and this limits the sophistication of the software. Until recently hardware was the only viable choice.

Computer simulation is a powerful tool in the Research environment. The time constraints imposed by real-time systems obviously do not exist during simulation. But accurately simulating the recording process can consume vast computer resources due to its complexity, and the fact that iterative techniques often need to be used. Computational performance therefore remains an important issue.

Parallel Processing promises computational performance of orders of magnitude greater than standard sequential architecture computers. What is unrealistically complex to implement currently in a real-time system (for example, large Neural Networks or Artificial Intelligence techniques) will be viable in the future. The computer industry is still at the bottom of the parallel processing 'learning curve', as is the magnetic recording industries use of it.

Occam and the ***transputer*** represent the first parallel processing language and hardware architecture designed for general purpose use. They reduce the architectural and semantic gaps between the algorithm and its implementation for parallel systems. But what benefits does this bring to digital magnetic recording ?

1.6. References for Chapter 1.

BLAHUT, R.E. Theory and Practice of Error Control Codes. Addison-Wesley Publishing Co. Inc. 1983.

COLE, G. Tape Leader ?, Electronics World, March 1991, p197.

DONNELLY, T., Mapps, D.J. & Wilson, R. High Density Data Storage on Audio Compact-cassette Tape using a Low Cost Tape Transport. I.E.R.E. Sixth Int. Conf. Video, Audio and Data Recording, Univ. of Sussex, March 1986.

DONNELLY, T., Mapps, D.J. & Wilson, R. An Intelligent Microprocessor Interface for a Low-cost Digital Magnetic Tape Recorder. Euromicro 87, Thirteenth Symposium on Microprocessing and Microprogramming, Portsmouth, September 1987.

DONNELLY, T. Real-Time Microprocessor Techniques for a Digital Multitrack Tape Recorder, Ph. D. Thesis, Polytechnic South West, 1989.

EMI, Modern Instrumentation Tape Recording: An Engineering Handbook. Thorn EMI. 1986.

FOX, B. Shots fired in the battle for DAT, Electronics Weekly, November 28 1990.

GALLAND, F.J.(Editor), Dictionary of Computing, John Wiley & Sons, Chichester, UK, 1982.

VAN GESTEL, W.J., Driessen, L.M.H.E. & Moeskops, J.C.F. A Multitrack Digital Audio Recorder for Consumer Applications. Journal of the Audio Engineering Society, Vol. 30, No. 12, December 1982.

HILL, R. A First Course in Coding Theory. Oxford University Press, New York, 1986.

HOCKNEY, R.W. & Jesshope, C.R. Parallel Computers 2: Architecture, Programming and Algorithms, IOP Publishing Ltd., Bristol , England, U. K., 1988.

INMOS Ltd., Occam 2 Reference Manual. Prentice Hall International (UK) Ltd., 1988 (a).

INMOS Ltd., Transputer Development System. Prentice Hall International (UK) Ltd., 1988 (b).

INMOS Ltd., Transputer Reference Manual. Prentice Hall International (UK) Ltd., 1988 (c).

JACOBY, G.V. A New Look-ahead Code for Increased Data Density. I.E.E.E. Transactions on Magnetics, Vol. MAG-13, No. 5, September 1977.

JORGENSEN, F. The Complete Handbook of Magnetic Recording. TAB Books Inc., USA, 1988.

LIN, S. & Costello, D.J. Error Control Coding: Fundamentals and Applications. Prentice-Hall Inc., 1983.

MACKINTOSH, N.D. The Choice of a Recording Code, Int. Conf. Video and Data Recording, Southampton, I.E.R.E. Conf. Proc. No. 43, 1979 (a).

MACKINTOSH, N.D. A Superposition-based Analysis of Pulse-Slimming Techniques for Digital Recording, Int. Conf. Video and Data Recording, Southampton, I.E.R.E. Conf. Proc. No. 43, 1979 (b).

MALLINSON, J.C. Recording Limitations, Chapter 5 of Magnetic Recording, Volume I: Technology. Series Editors C.D. Mee & E.D. Daniel. McGraw-Hill, Inc. USA, 1987 (a).

MALLINSON, J.C. The Foundations of Magnetic Recording, Academic Press, Inc. (London) Ltd., 1987 (b).

NOTTLEY, G.C. 3-Position Modulation (3PM): A Technical Appraisal. The Fourth International Conference on Video and Data Recording, Univ. Southampton, April 1982.

ONISHI, K., Ido, K., Inoue, T., Inaga, M. & Tanaka, K. Consumer use Compact Cassette Digital Audio Recorder, Pre-print No. 2092 (H3), 75th. Convention of the A.E.S., Paris, March 1984.

PATEL, A.M. Adaptive Cross Parity (AXP) Code for a High-Density Magnetic Tape System. IBM Journal of Research and Development, Vol. 29, No. 6, November 1985.

POULIART, W.H.P. & Vandevenne, J.P.H. Electrical Intelligence Storage Arrangement. U.S. Patent No. 2807004, September 1957.

SAKOMOTO, N., Kogure, T., Kitagawa, H. & Shimada, T. On High-Density Recording of the Compact-Cassette Digital Recorder, J. Audio Eng. Soc., Vol. 32, No. 9, September 1984.

SANDLER, M.B., Hayat L., Costa L. & Naqvi A. A Comparative Evaluation of DSPs, Microprocessors and the Transputer for Image Processing, I.E.E.E., 14th Int. Conf. Acoustics, Speech, and Signal Processing, 1989.

SCHOUHAMER-IMMINK, K.A. Coding Techniques for Digital Recorders, Prentice Hall International (UK) Ltd, 1991.

SMITH, O. Some possible forms of Phonograph, The Electrical World, 9, 161, 1888.

WATKINSON, J. The Art of Digital Audio, Focal Press, England, 1989.

2. Experimental Apparatus.

2.1. Overview.

The experimental apparatus shown in figure 2.1. provides an environment in which digital data may be recorded onto a compact-cassette tape, and subsequently replayed and analysed. Part of the channel is implemented in software running on a ***transputer***. The ***transputer***'s host computer, an IBM Personal Computer model XT, (referred to from now on as the IBM PC) was also used to store results and provide an interface between the ***transputer*** and tape transport mechanism.

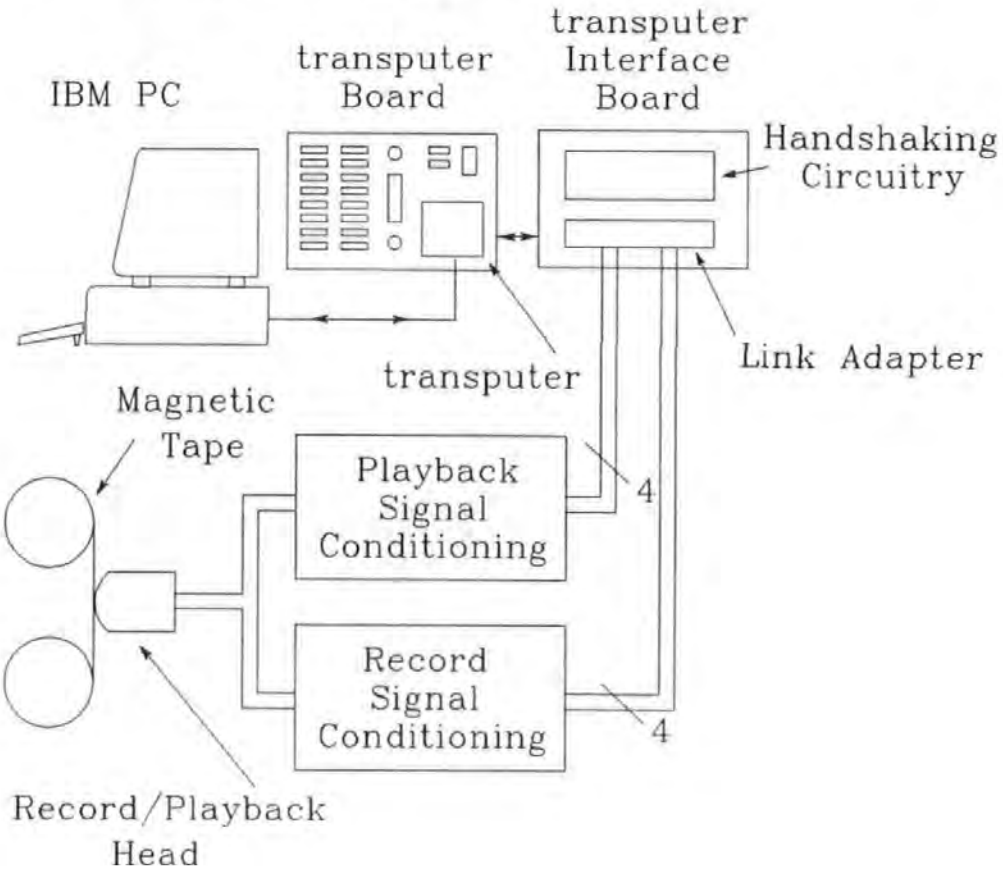


Fig. 2.1. The Experimental Apparatus.

Figure 2.2 is a photograph of the experimental apparatus (PC not shown). On the right is the compact-cassette tape transport

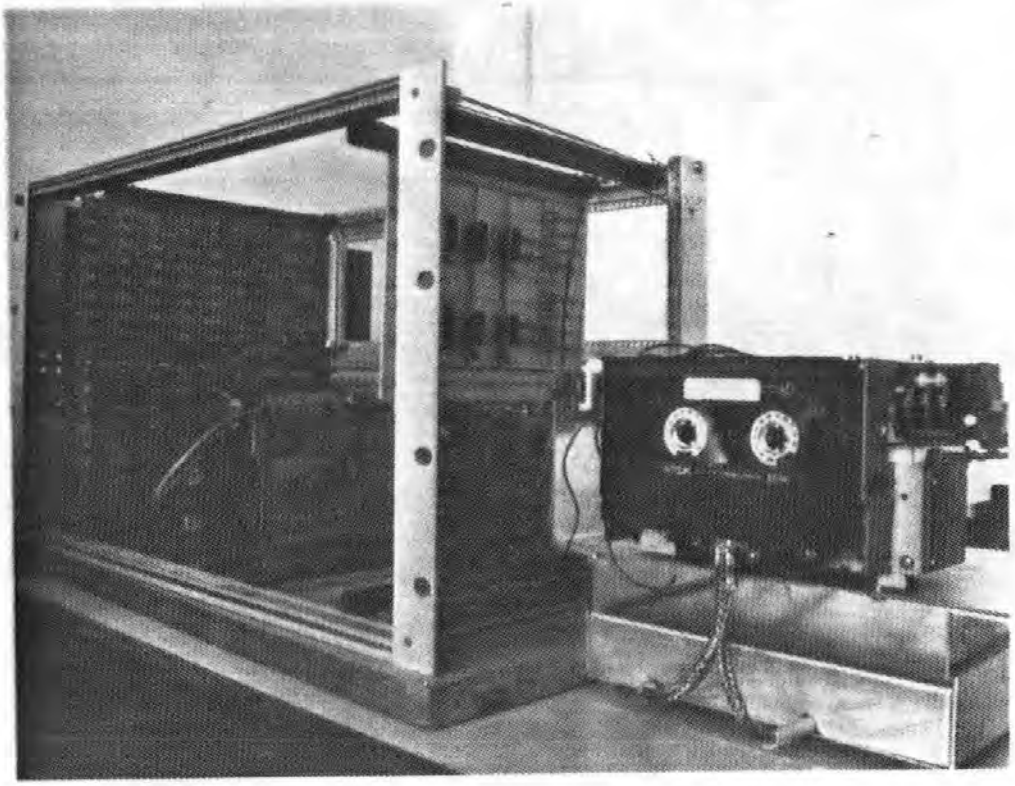


Fig. 2.2. Photograph of Experimental Apparatus.
(Original in Colour)

mechanism. The five circuit boards in the card-cage to the left are (left-to-right):

i) ***transputer*** board.

ii) Link Adapter Interface Card.

iii) Solenoid Control Board.

iv) Write Amplifier board.

v) Gated Cross-Over Board.

The Read Amplifier Boards are housed in the aluminium case below the tape mechanism. Circuit diagrams for the above can be found in Appendix E.

2.2. The *transputer* and its Development System.

2.2.1. The INMOS *transputer*.

Transputers are self contained microcomputers designed to execute ***occam*** processes efficiently. ***Transputers*** are highly integrated devices, fabricated on a single piece of Silicon. Figure 2.3 shows the functional block diagram for a typical ***transputer***. As well as supporting features common to standard microprocessors, ***transputers*** also provide hardware support for concurrency and inter-process communication. ***Transputers*** also incorporate much of the external circuitry needed to support conventional processors,

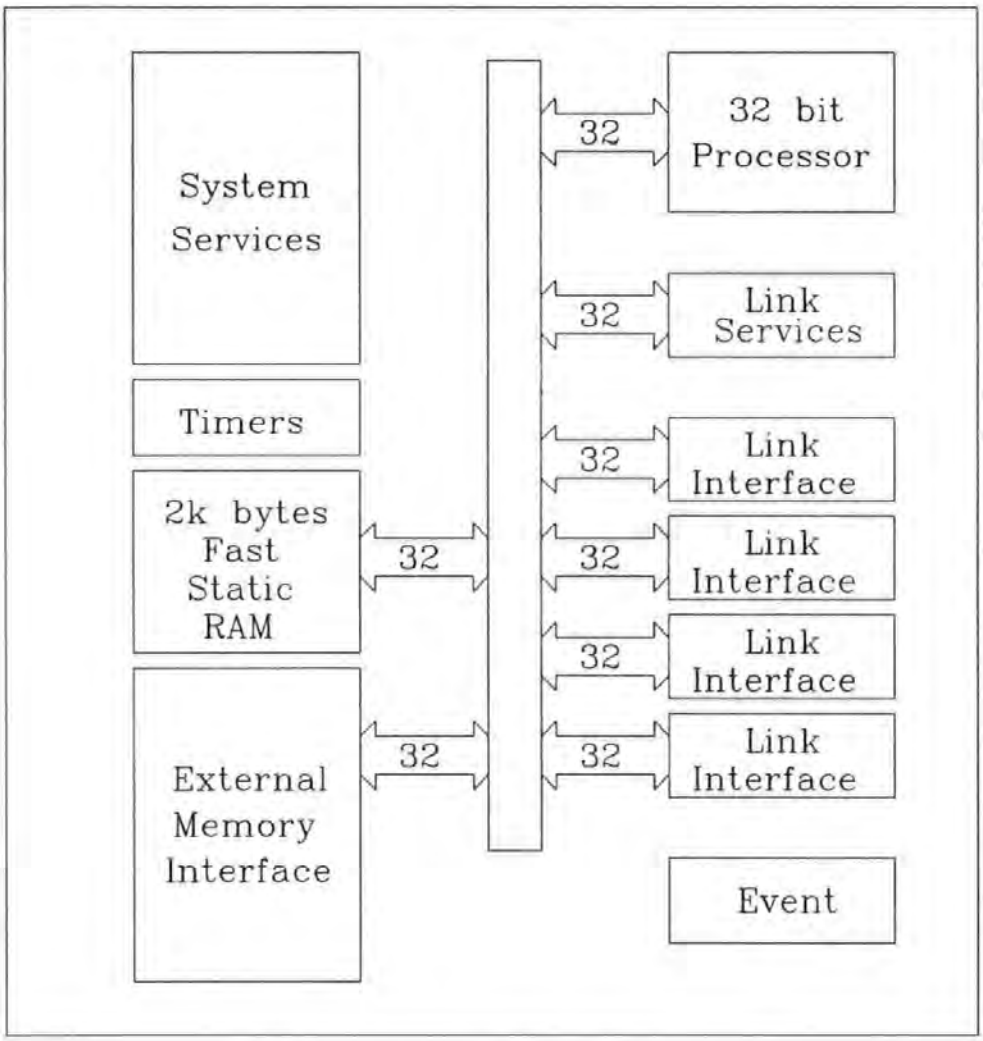


Fig. 2.3. ***transputer*** Block Diagram.

significantly easing the design of multiple processor networks.

Key features of ***transputers*** are (INMOS, 1988(a)):

i) RISC-Type Central Processing Unit.

The design of the Central Processor Unit (CPU) at the heart of the ***transputer*** was influenced by RISC (Reduced Instruction Set Computer) ideology. The first noticeable influence is the simplicity of many of the instructions. A simple or reduced instruction set can be implemented with fewer transistors. This in turn allows faster clock rates, resulting in higher computational performance. This philosophy does not restrict the use of complex expressions, as these may be assembled from simpler ones by the compiler (INMOS, 1988(b)). As the CPU may be implemented with a small number of transistors it occupies less space on the integrated circuit. This was a key factor in the design of the ***transputer*** as the space saved was used to implement the non-standard features mentioned above.

Transputers are Stack oriented processors, unlike most modern microprocessors that are register oriented. ***Transputers*** store all variables in memory, loading them onto the 3 stage stack only for evaluation. This style of architecture allows the ***transputer*** to switch from process to process very quickly (in less than 1μS) as very little internal state needs to be saved (i.e. contents of registers transferred to external memory).

Stack oriented architectures can hinder computational performance as external memory references are usually slower than register references. To compensate for this, each ***transputer*** has some very fast Static RAM that can be accessed in a single cycle (nearly as quickly as a register). The IMS T414-15 (for example) has 2K bytes of memory with an access time of 67nS (this can be thought of as five hundred 32 bit registers).

ii) Process Scheduler.

The microcoded process scheduler controls the sharing or time-interleaving of the CPU between processes for concurrent execution. The following is a simplified explanation of how this concurrency is implemented. When several processes are being executed concurrently, two lists are maintained: one lists processes that may

proceed immediately (the Active list), the second lists processes that are waiting because of a communication that cannot proceed (the Inactive list). The CPU executes the process at the top of the Active list until a message communication cannot proceed. This situation will arise if the process attempts to output a message to a process that is not ready to accept it, or if the process attempts to input a message that has not been sent. Under control of the process scheduler, the process is de-scheduled (i.e. the CPU stops executing it) and put on the Inactive process list, and the process now at the top of the Active list is scheduled. The scheduler moves processes from the Inactive list to the Active list when the communication can proceed. In an attempt to share-out the CPU's time evenly between processes, a process is de-scheduled and moved to the bottom of the Active list after a certain amount of time (its time-slice period). Section 3.2.2.1. develops these ideas using the data acquisition process as a vehicle.

Message passing between processes or its control can be seen to be central to the ***transputer***'s implementation of concurrency. The rules governing inter-process communication were developed by Hoare for the mathematical language CSP (Communicating Sequential Processes (Hoare, 1985)). ***Occam*** may therefore be viewed as a practical implementation of the CSP.

Controlling the process scheduling in hardware is desirable for several reasons:

a) The concurrency is transparent to the programmer. The corollary of this is the programmer does not know when any particular process is being executed. This can cause considerable problems in a real-time system (as is this system, see section 3.2.2.1).

b) The programmer does not need to write a process scheduler. This not only lessens the number of tasks the programmer needs to do, it also means the CPU is relieved of the extra processing involved in executing this code.

c) Process switches are carried out by hardware and therefore take very little time (less than 1μS to de-schedule one process

and schedule the next), a very important feature for a processor that may be concurrently processing hundreds of processes.

iii) Communication Links.

Processes constructed to run in parallel communicate via Channels. If processes are running on the same *transputer*, channel communication takes place via an internal memory location (a soft channel). If they are running on separate *transputers*, a Link (a hard channel) is used.

Each Link is an asynchronous, autonomous, Direct Memory Access (or DMA) engine. Each Link can bi-directionally transfer data at rates up to 10 Mbits per second in each direction. As the Links operate autonomously, once the communication has been initiated, the processor plays no further part in the transfer, freeing it to proceed with other tasks.

Transputers have a high maximum instruction throughput and a wide I/O bandwidth, for example 10 million instructions per second and 80Mbits per second for a 20MHz IMS T414. This makes them well suited to the evaluation of complex algorithms at high data rates. *Occam* includes a Timer type that represents the current state of a clock when read. *Transputers* directly support this model of time by providing two hardware timers, with periods of 1μS and 64μS.

2.2.2. The Software Development and Run-Time Environment.

The *transputer* Development System (TDS) (INMOS, 1988(c)) is a software package that supports the development of *occam* programmes. Programmes may also be run from within this environment. The TDS's software is split between a *transputer* board and host computer (the IBM PC in this case).

2.2.2.1. The *transputer* Board.

The *transputer* board (a Sension JD002, an INMOS IMS B004 equivalent) was populated with a 15MHz IMS T414 *transputer* (INMOS, 1987(a)), 1 Megabyte of RAM, together with various support circuitry (e.g. clock signal generation, memory configuration, electrical buffering e.t.c.). One of the *transputer*'s four Links is dedicated to communication with the IBM PC, via an interface card that plugs into one of the IBM PC's peripheral slots. This Link Interface board is bi-directional, converting between the eight bit parallel data format of the IBM PC and the serial format of the *transputer*'s Links.

2.2.2.2. Development System Software.

An Editor, *occam* Compiler and post-mortem Debugger, as well as several software Tools that assist the development and implementation of *transputer* based systems, are included in the TDS. These run on the *transputer*. The *transputer* board does not have direct access to any peripherals, such as a keyboard or screen. Instead, it uses those of the IBM PC. The host computer runs a programme, called the Server, that provides file input and output, screen display and keyboard input for the *transputer*.

A virtue of the TDS running on a *transputer* is that the same *transputer* may also be used to run and test the developed code. Although the network thus formed consists of only a single *transputer*, it does allow the code to be run and fully logically debugged.

2.2.3. Link Adapter Interface Board.

This board converts signals to and from *transputer* serial Link protocol and the parallel data format of the multiple-track compact-cassette recorder. When writing data to the cassette it

converts from the Link's serial format to a parallel format suitable for driving up to 8 Write Amplifiers. When data is to be read, it converts up to 8 asynchronous data channels to a format suitable for transmission down a single Link.

An INMOS IMS C011 Link Adapter (INMOS, 1987(b)) performs the basic serial to parallel and parallel to serial conversion. In Mode 1 this device converts the two wire bidirectional serial Link into two, hand-shaken, byte-wide parallel interfaces. Four hand-shake lines (two for input and two for output) enable the Link Adapter to comply with *occam*'s synchronised channel communication protocol.

When data are output from the *transputer* to drive the Write Amplifiers no synchronisation is required. The two output hand-shaking lines (QValid and QAcknowledge) are therefore connected together, effectively disabling output hand-shaking. Data received by the Link Adapter are therefore transferred to the output interface immediately (i.e. as soon as the data are 'Valid', they are 'Acknowledged'). A Bus Transceiver (an SN 74LS245) is used to drive the Write Amplifiers.

Occam Channels, and therefore *transputer* Links, are data or Event driven. The data driven aspects were preserved in the operation of the Link Adapter Interface board (as opposed to using a software polling technique). A dedicated circuit was designed (Jackson et al., 1989) to monitor the inputs for new data. When new data are detected they are latched, and the correct sequence of hand-shaking signals are generated to transfer the data to the *transputer*. Figure 2.4. shows the block and timing diagram of the input circuitry.

When the magnitude comparator (an SN 74LS688) detects new data it takes IVal high. This disables the transparent latch (an SN 74LS373) stopping further new data appearing at the comparator. When the Link Adapter has read the data from the flip-flops (an SN 74LS374) it takes IAck high. This clocks the flip-flops, transferring the new data to the input of the interface for the next read, making the P and Q inputs equal. The comparator takes IVal low to complete the cycle. The circuit reliably multiplexes up to 8 asynchronous data channels down a single Link, without losing data that arrives whilst

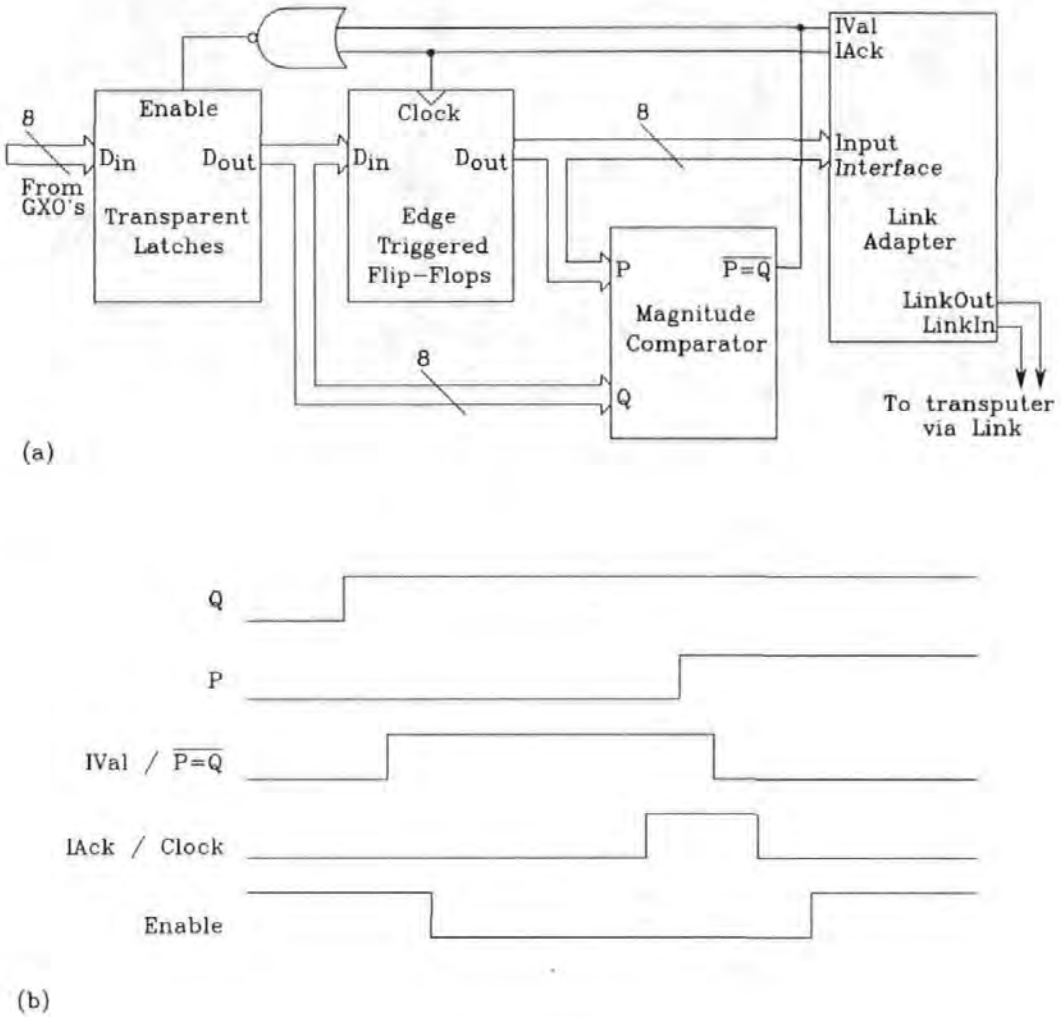


Fig. 2.4. Link Interface (a) Block Diagram,
(b) Timing Diagram.

the Link Adapter is in the middle of a data transfer.

A memory mapped port could have been used for this interface. However this would have required a software polling technique that would have consumed CPU time. Also, the port would need to be sampled approximately 58 times per 5kbps bit period to provide the same timing resolution (3.5μS) as that of the event driven interface. As all four tracks of Bi-Phase-L encoded data generate on average only 6 events during the same bit period, the communication bandwidth of the acquisition process was reduced by a factor of approximately 9.5.

2.3. Signal Conditioning.

The TTL voltage levels of the digital data to be recorded need to be converted into an analogue current signal, capable of driving the record head. The Write Amplifier performs this function. On replay, the amplitude of the voltage waveforms are very low (approximately 1.75mV peak-to-peak when replaying a Bi-Phase-L encoded PRBS at 5kbps), and so low noise Head Amplifiers were built to amplify them.

Section 1.3.1. introduced the idea of a compromise between the flexibilty facilitated by the use of software, and the higher data rates facilitated by hardware. The chosen solution was to use hardware up to and including the detection system (although this was modified for one part of the investigation). A Gated Cross-Over Detector (GXO) (Whatton, 1973) was built to perform this task.

The analogue electronics was found to suffer from high levels of electromagnetically induced noise. The noise sources were found to be the ***transputer*** and IBM PC screen and switched mode power supply. As these items could not be turned off during testing a number of preventative measures were taken:

- A separate linear power supply was used for the analogue electronics.
- Screened, twisted-pair wiring was used between the head and head amplifiers, which in turn were housed in a separate metal enclosure.
- A single earth return point (Star point) was used. This was situated physically and electrically as close to the head (i.e. the most noise critical section of the circuit) as possible.

Although these measures reduced the noise, the residual noise was still significantly higher than the magnetic medium noise, and therefore a limiting factor on the performance of the system.

2.3.1. Write Amplifier.

The data to be recorded were output from the *transputer* as TTL logic levels. It is the Write Amplifiers function to control the flow of current through the recording head so that the magnetic field imprinted in the tape represents these logic levels. The representation used was saturation recording, where a logic '0' results in the tape being saturated in one direction, and a logic '1' results in the tape being saturated in the opposite direction.

To record at high linear data rates, the transition between saturated magnetic regions must be short, and therefore the reversal of current through the head must be rapid. This not only enables high packing densities to be recorded, but also produces a high rate of change of magnetic field at playback, and a correspondingly large induced voltage.

Inductors oppose any change in current passing through them. The inductive record head used therefore opposes the rapid change in current desired. If driven by a simple resistive source the change in current would be exponential. Constant current sources were therefore chosen to drive the inductive head (resulting in a near linear increase in current).

A bridge configuration was chosen, see figure 2.5(a). The transistors were operated in pairs, T1 with T2, and T3 with T4. The diagram shows T1 and T2 fully conducting, enabling current to flow in one direction through the head. The current is reversed by removing the base drive from T1 and T2, and driving T3 and T4 into saturation.

The performance of the write amplifier was simulated using a commercial simulation package, MSpice from Mentor Graphics, running on an Apollo Graphics workstation. Figure 2.6 shows the results of the analysis. The current waveform of Graph (a) shows the desired rapid current reversal. However, Graph (b) reveals that the switching transistors sustain a voltage spike in excess of 200 V between their emitter and collector terminals, due to the heads inductance and the fast switching times.

As this voltage far exceeds the transistors quoted maximum of 40V, protection diodes were placed between their collector and

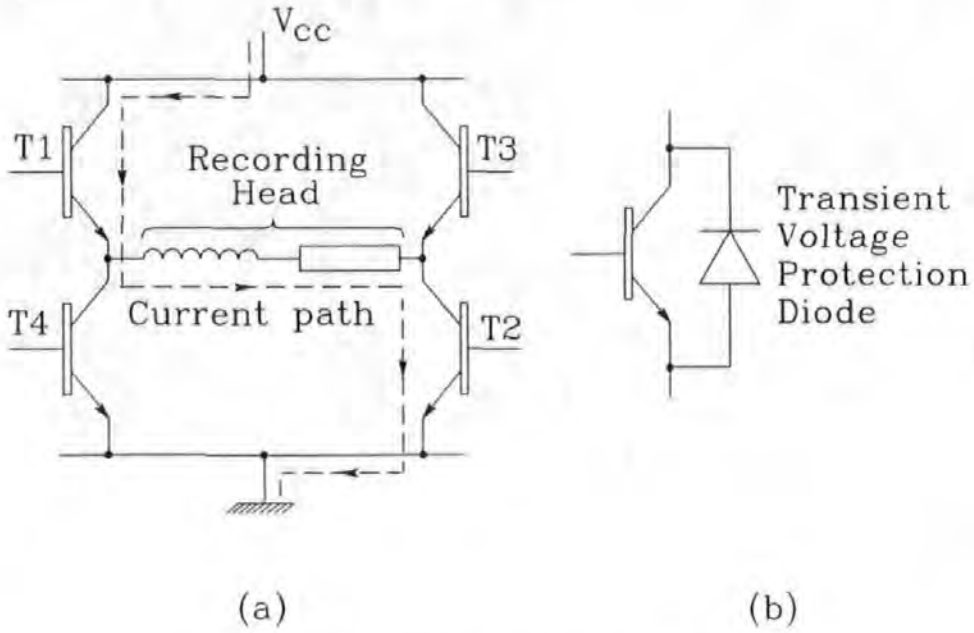


Fig. 2.5.(a) Write Amplifier Bridge Circuit.
(b) Diode Protection of Transistors.

emitter terminals, see figure 2.5(b). The diodes introduced a low impedance current path for the induced voltage spike, bypassing, and thereby protecting, the transistors.

Unfortunately, this degrades the overall performance of the circuit. The diodes not only present a low impedance path to the unwanted induced voltage, but also to the current that should be flowing through the head. Consequently, for the time the diodes were conducting (i.e., whenever the induced voltage is greater than approximately 0.6V); the current through the head did not increase as quickly as without the diodes, see figures 2.6 (c) and (d). Though compromised, the Write Amplifier still had a rise-time of 19μS with a write current of 0.33mA.

2.3.2. Read Amplifier.

The output from the 4-track inductive head, whilst replaying a Bi-Phase-L encoded 5KHz PRBS, is just 1.75mV peak to peak. The first stage of amplification was therefore placed physically as close as

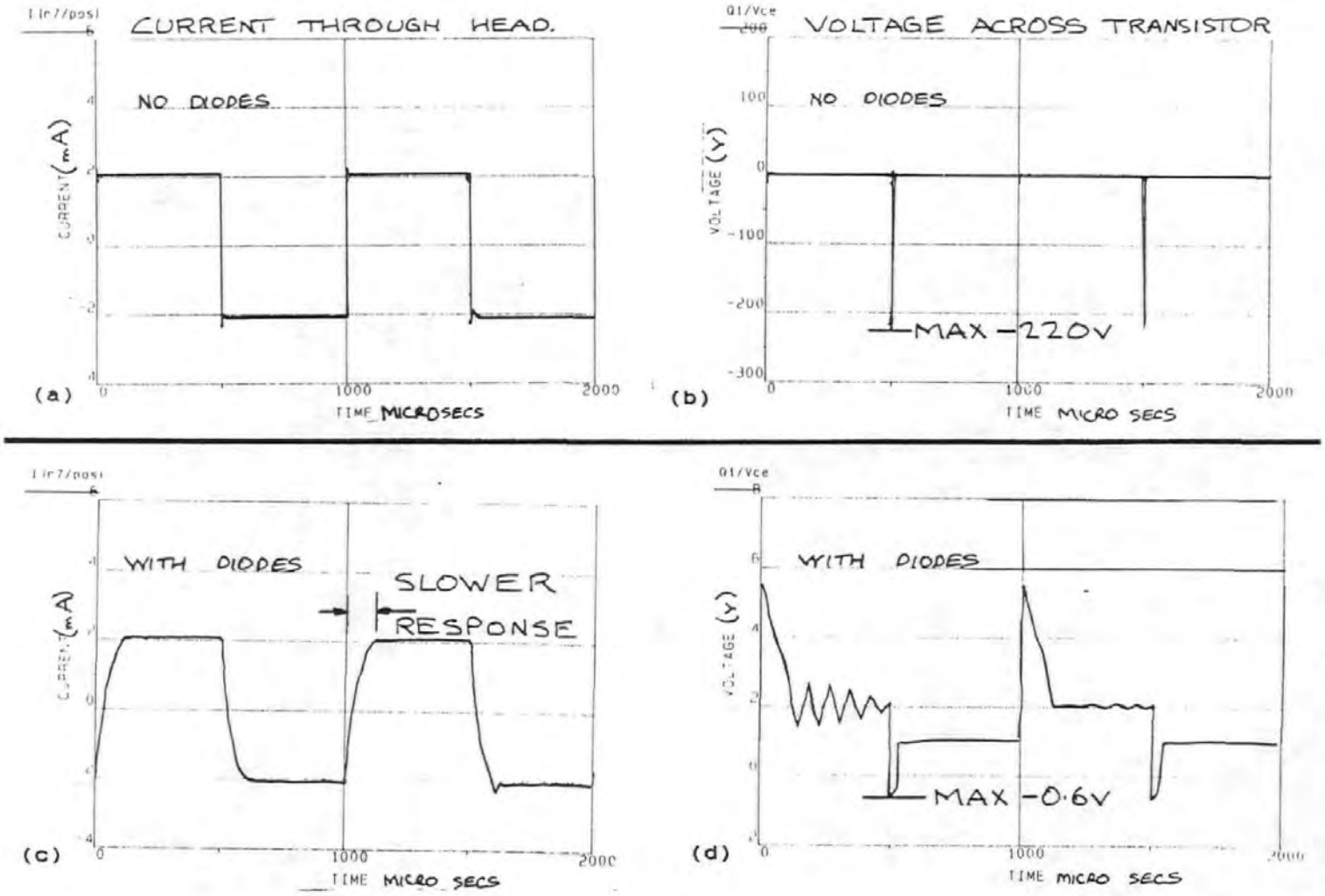


Fig. 2.6. Bridge Circuit Simulation.

possible to the head to keep connector lengths as short as possible.The connectors were screened twisted pairs. This kept electro-magnetic pick-up to a minimum, preserving the signal to noise ratio.

Although a 4-track inductive head was to be used initially, the amplifier was designed to have sufficient gain for it to be able to cope with heads that produce even lower voltages. Figure 2.7 shows the Read Amplifier block diagram, whilst the following points provide more detail.

i) Standard operational amplifiers were used (a low noise version of the NE5534) to save time rather than designing a discrete transistor circuit, even though a discrete circuit would probably produce the best performance.

ii) Two gain blocks were used, as any single operational amplifier that met the specification would be expensive. Hardware costs per track are an important consideration in a multiple-track system.

iii) A solenoid activated 4-pole change-over relay was used used to switch the head between the read and write amplifier. This was mounted close to the head on the read amplifier board.

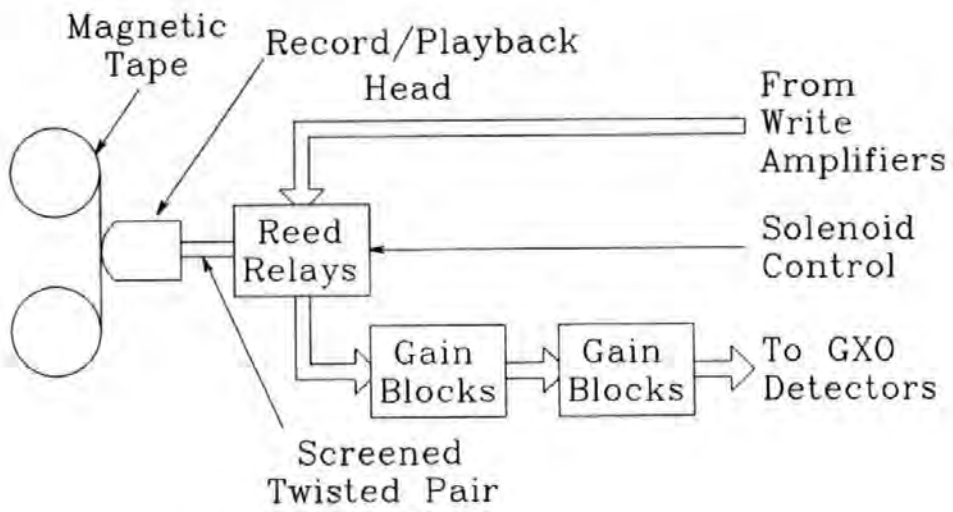


Fig. 2.7. Read Amplifier Board Block Diagram.

iv) An earthed ground plane was fabricated on the component side of the circuit board to reduce electromagnetically induced noise.

v) Low value resistors were used to reduce Johnson noise (proportional to resistance) and noise produced by induced currents (simply from Ohms Law).

2.3.3. Gated Cross-over Detector.

The analogue signal from the head amplifier must be converted into a digital one before it can be input to the ***transputer***. There are many ways of doing this, each with their advantages and disadvantages (Mackintosh, 1979). The use of a linear quantising ADC (Analogue to Digital Converter) was rejected for the compact-cassette system as it places a high computational demand on the software, see section 1.3.1 (although one was used for a specific part of the investigation, see section 3.4). A Gated Cross-Over circuit was chosen as it can achieve a high level of performance, and is simple in design. Its popular use would also enable straightforward comparisons to be made with other systems.

The GXO produces two intermediate signals; the first referred to as the Gating signal indicates pulse peak-centres, the second referred to as the Polarity signal indicates the polarity of the peak. The Polarity signal is transferred to the output of a D-type flip-flop by the Gating signal, see figure 2.8.

i) Gating Signal.

Pulse peak-centres are determined by differentiating the signal, to give a 90 degree phase shift, and then passing it through a zero crossing detector. An active differentiator (based on an LM747 operational amplifier) was chosen because of its superior performance compared to a passive one over a wide range of frequencies. The zero-crossing detector is a comparator (an LM 393N) with hysteresis. The hysteresis was set at 40mV. Each zero-crossing edge was then

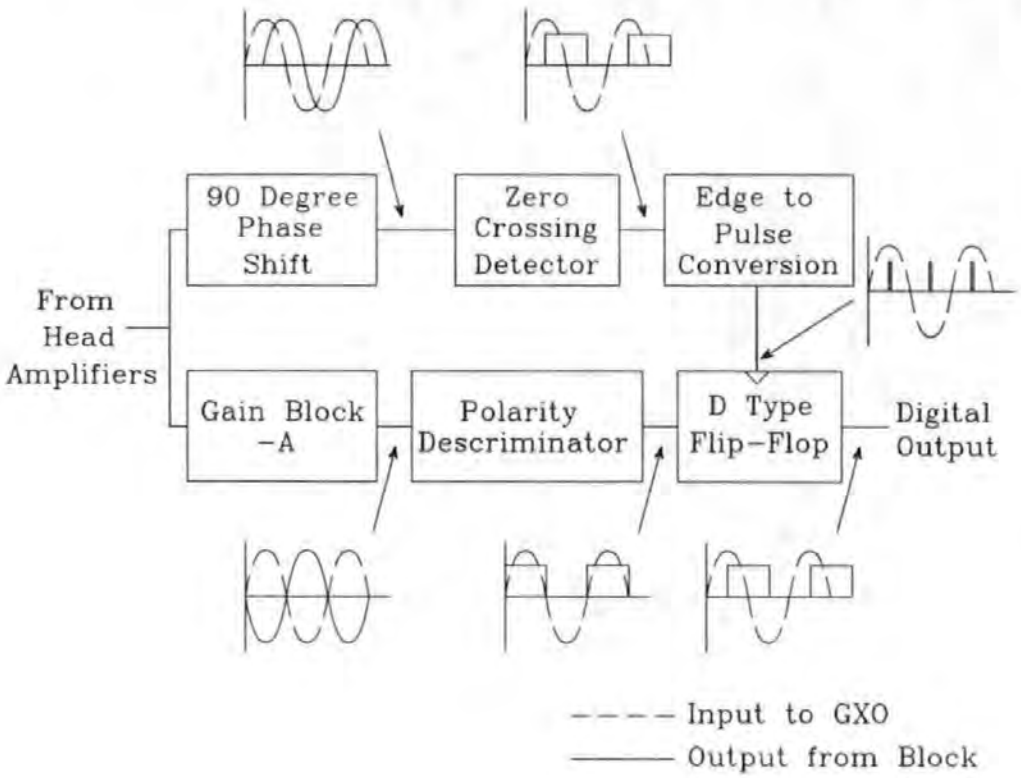


Fig. 2.8. Block diagram of the Gated Cross-Over. Detection Circuit.

converted into a gating pulse (approximately 0.5mS wide) by exclusively ORing it with a delayed version of itself.

ii) Polarity Signal.

Following a gain block, a comparator (an LM 393N) with variable hysteresis (set at 0.8mV) discriminates the polarity of the peak. It is a feature of the GXO detector that such a low value of hysteresis may be used. Noise may produce false triggering in the Polarity signal, but will not be gated to the output of the GXO as the Gating signal hysteresis is set much higher.

iii) Digital output.

A D-Type flip-flop (an SN 74LS74) outputs the polarity signal on the positive edge of the Gating signal, i.e. at the centre of the pulse.

A more detailed description of the operation and optimization of GXO detectors is given by Mackintosh (Mackintosh, 1979).

2.4. The Tape Transport Mechanism and its Control.

The compact-cassette tape transport mechanism was solenoid-controlled. Due to the evolutionary development of the project, these solenoids were controlled by the IBM PC via a general purpose interface card and solenoid drive amplifiers. The ***transputer*** therefore sent messages to the IBM PC, that in turn controlled the mechanism.

The situation was further complicated as the IBM PC was already running an application programme (the TDS Server). The simplest solution was to install a number of interrupt service routines before starting the Server. Each interrupt service routine, written as a DOS Terminate and Stay Resident programme in assembly language (Scanlon, 1985), provided a basic cassette transport function (e.g. fast forward, play, e.t.c.). To control the tape transport mechanism, the ***transputer*** sent the relevant interrupt number to the Server, which generated the required interrupt. The interrupt service routine generated the signals, which, via the IBM PC's Interface Card and Solenoid Drive Card, energised or de-energised the relevant tape transport mechanism solenoid.

2.4.1. The IBM PC Interface Card.

The functionality of the board can be divided into five areas, see figure 2.9.

i) Buffering to the Computer.

To lessen the possibility of an external electrical fault damaging the IBM PC, the computer's address bus, data bus and control signals were buffered (using two SN 74LS244s and an SN 74LS245).

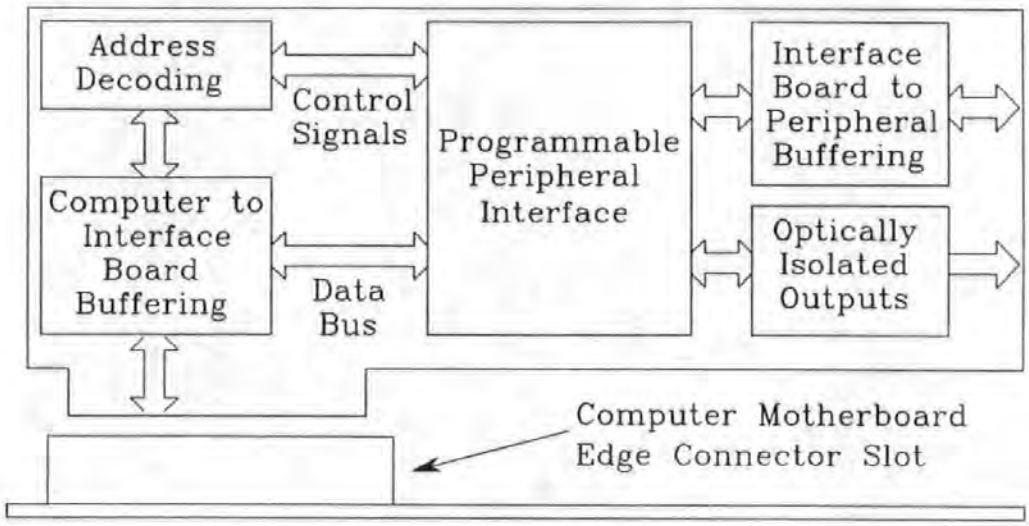


Fig. 2.9. IBM PC Interface Card.

ii) Address Decoding.

The board was mapped into the IBM PC's I/O space. Unfortunately, the space in the I/O map reserved for Prototype boards at Hex 300-31F (IBM, 1986) is used by the TDS Link Adapter board. It was therefore mapped in at Hex 360-36F, an area not used during this project. Although only one peripheral interface IC was used, the revised address decoding scheme provided sufficient 'Chip Select' lines to allow up to four such IC's to be connected without modification.

iii) The Peripheral Interface IC.

An Intel iAPX 8255A Programmable Peripheral Interface (PPI) integrated circuit (Intel, 1987) was selected as it was designed to be fully compatible with the IBM PC's microprocessor, and exceeded the required I/O specification.

iv) Output Buffering.

The PPI is fabricated using CMOS technology and is therefore not suited to driving long lines or low impedances. Therefore 21 lines of the PPI were buffered with TTL transceivers (SN 74LS245). This also provides some further electrical protection for the

computer.

v) Optically Isolated Outputs.

The voltages and currents associated with the solenoids are potentially very damaging to logic circuits. Therefore the three lines used to drive the Solenoid Drive Card were optically isolated (using an ILQ74). This provided a very high level of protection.

2.4.2. The Compact-Cassette Tape Transport Mechanism.

This is a standard solenoid controlled compact-cassette tape-transport mechanism (an SF925F, available from Hart Electronics, Hertfordshire), with a modified head-mount and tape-guide. In a standard mechanism the head is secured by two screws that locate in two pillars, the height of one of which may be varied a small amount. This allows the azimuth of the head to be adjusted. In the modified tape transport mechanism, two new adjustable mounting pillars were machined, see figure 2.10.

This new mounting assembly allowed the head to be moved across the face of the magnetic tape simulating lateral head displacement or to be rotated simulating azimuth skew. Measured displacements and skews greater than 1.2mm and 2.7 degrees respectively could be thus introduced. Two bolts were used to adjust the heights of the mounting pillars. Each bolt had a screw pitch of 0.4mm. The screw-driver used to make the adjustments had a pointer (16cm long) attached to it. A circular dial, graduated in degrees, was used to measure the angle of rotation to an accuracy of approximately 1 degree, corresponding to a change in pillar height of 1.1×10^{-9} metres.

Tape guides are normally mounted on the side of the head. This would physically distort the tape during head displacement and skew tests. By mounting the tape-guide on a separate fixed height pillar, the transport of the tape past the head was unaffected by the movement of the head.

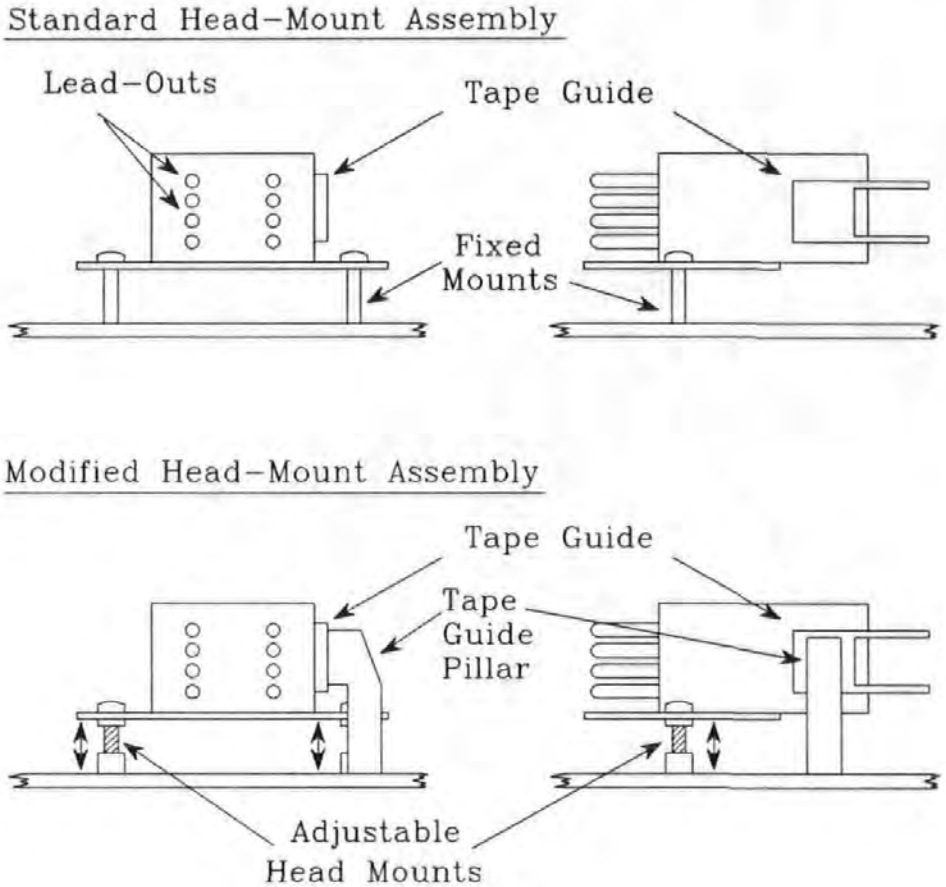


Fig. 2.10. Modified Head Mount.

The operational mode of the tape transport mechanism depends on the energised state of two solenoids;

Solenoid.		Mode
1	2	
energised	energised	play
energised	-	fast-forward
-	energised	fast-rewind
-	-	stationary

The solenoids require an initial current of 500mA, which may be reduced to a holding current of 250mA once energised.

2.4.3. Solenoid Drive Card

The Solenoid Drive card inputs the TTL voltages from the IBM PC interface card and converts them to the higher voltages and currents required to energise the solenoids. The circuit (Donnelly, 1986) shown in figure 2.11, connects one end of the solenoid's coil to +12v whilst the other is connected via two FETs to 0v and -12v. To produce the initial current of 500mA, both FETs are turned on, connecting the solenoid's coil between +12 V and -12 V. The voltage applied to the gate of one of the FETs is controlled by a resister/capacitor timing circuit. After approximately 0.1 seconds, the gate voltage drops below the turn-on threshold, and the FET turns off. This leaves the solenoid's coil connected between +12v and 0v through the remaining FET, maintaining the required holding current of 250mA.

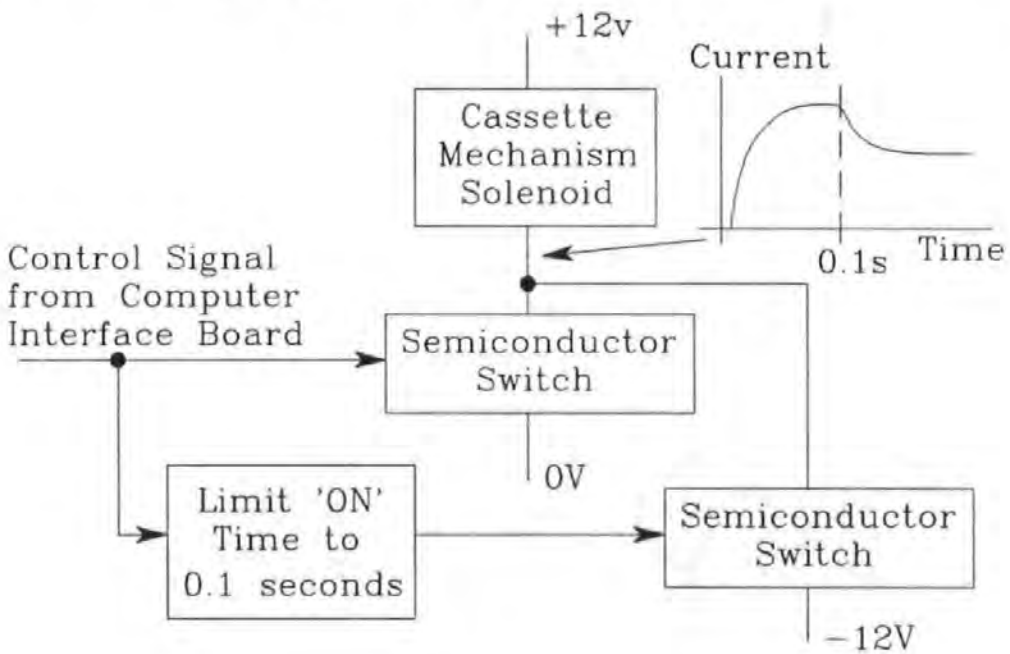


Fig. 2.11. Solenoid Control Circuit.

2.4.4. Record and Replay Heads.

The majority of the experimental work was carried out using a commercial 4-Track, 4-Channel audio-frequency inductive head (type

HQ551, available from Hart Electronics, Hertfordshire). The only modification made to it was the removal of its tape guide (for use with the modified head mount). Its specification can be found in Appendix F. The same head was used for record and replay.

In an allied research project Thin-Film techniques are being used to fabricate heads. The components of the head are fabricated by sputtering substances onto a substrate and then (using photolithography) etching away the material not required. The aim is to produce a head with 18 tracks across the same width as the standard compact-cassette head. The track format is shown in Figure 2.12.

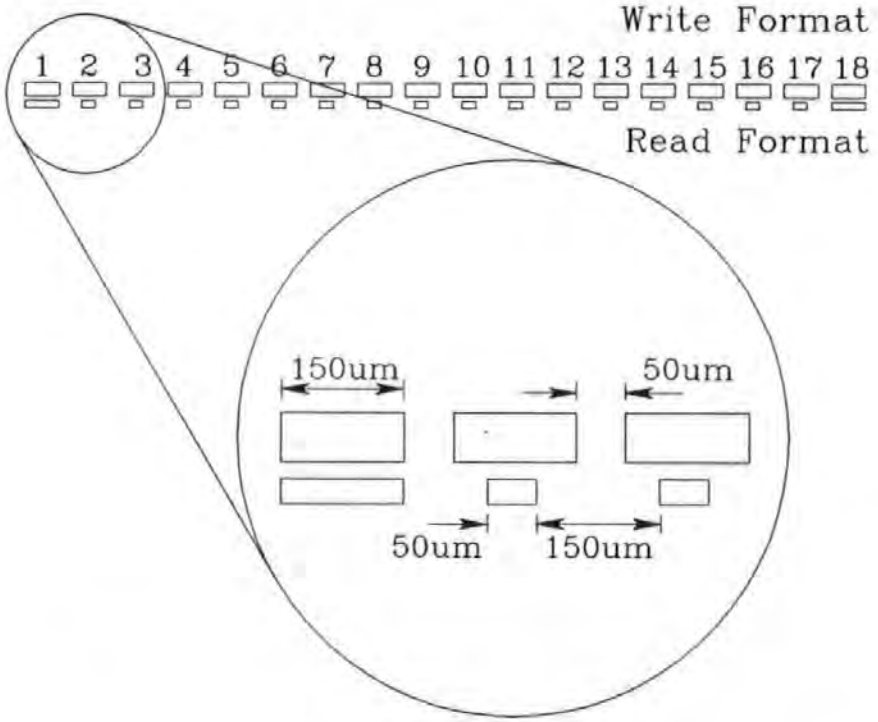


Fig. 2.12 Track Format of the 18 Track Thin-Film Head.

The 18 track head is an inductive write, Magneto-Resistive (MR) read design. The write magnetic circuit is formed by two Permalloy pole pieces with a single turn copper coil between them. The MR read element is a stripe of Nickel Iron (in the proportion 80% to 20%) 40nm thick, 40μm high, fabricated between the two pole pieces. This MR stripe gives approximately a 2% change in resistance

depending on the direction and strength of the sensed magnetic field. The central 16 MR read elements (or more specifically the flux-guides) are 50μm wide, whilst the two outermost tracks are 150μm (the full written track width). It was envisaged that these two edge tracks would be used in some way to compensate for track misregistration.

Unfortunately a complete fabrication of the head was not achieved in time for it to be used in this project. However, the read elements and associated lead-out were finished. This allowed data recorded by the inductive head to be replayed using the MR head. Figure 2.13 shows a micrograph of 7 of the 18 read elements and lead-outs.

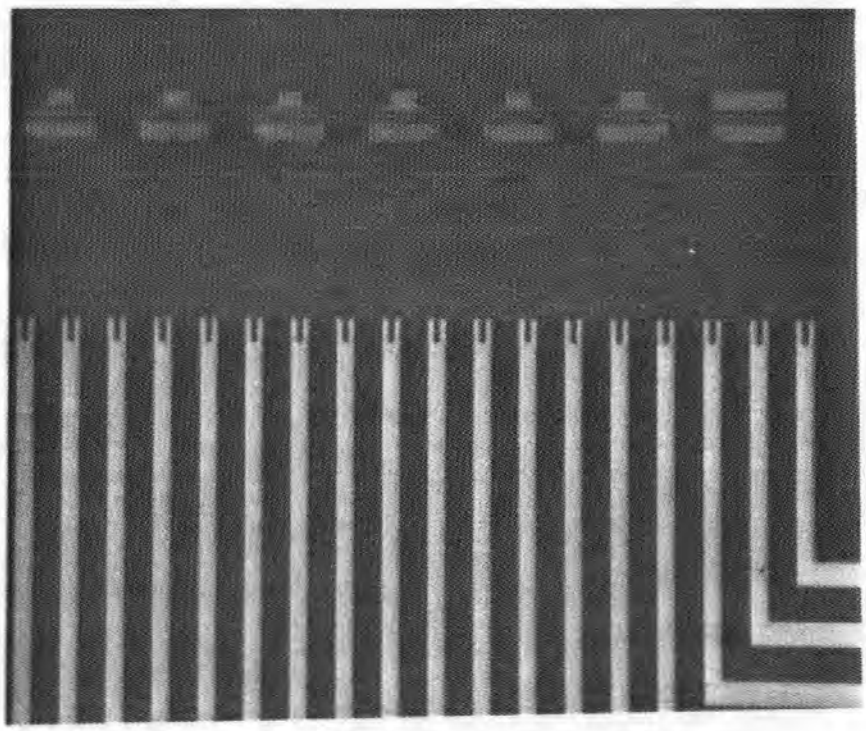


Fig. 2.13. MR Read Elements.
(Original in Colour)

The write pole-pieces were designed to perform a secondary role of providing a bias field for the MR element. As these were not

present, a small permanent magnet was used instead.

The MR stripe does not produce a voltage in response to a change of flux (as does an inductive head). Instead, its resistance changes. This change is detected by passing a constant current through it (2mA was used) and monitoring the change in voltage. As the change in resistance is only 1 or 2 ohms and the lead-out resistance is approximately 160 ohms a bridge circuit was used to remove this offset. The MR stripe formed one arm of the bridge, as shown in figure 2.14.

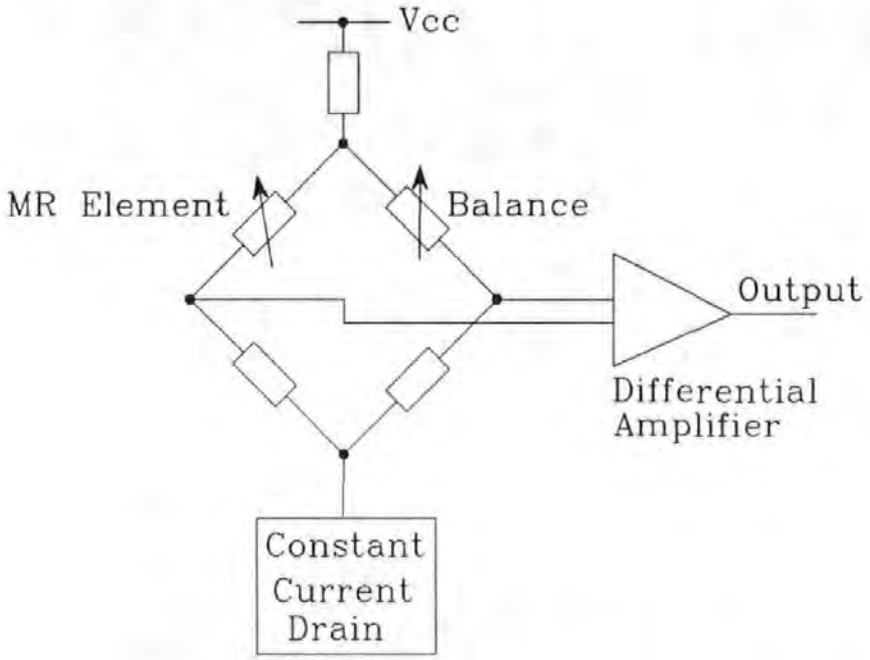


Fig. 2.14. Use of Bridge to detect change
in MR element Resistance.

2.5. Summary.

The investigation required a multiple-track magnetic tape system that would permit the recording and replay of digital data, and allow various error causing mechanisms to be introduced in a measured way. Much of the signal processing was to be carried out in software using parallel processing techniques. A system based on a

compact-cassette mechanism was designed to meet these requirement.

The signal conditioning circuitry required for the recording and replay of digital data was built. This included the interfacing circuitry required to integrate a ***transputer*** into the data channel thus formed. The compact-cassette tape transport mechanism's head mounting arrangements were modified to allow measured amounts of head-to-tape misregistration and azimuth skew to be introduced. The electronics allowed both conventional 'inductive-write, inductive-read' as well as 'inductive-write, magneto-resistive-read' modes of operation to be investigated.

The write amplifier was designed to produce a rapid reversal of current through the inductive head, permitting recordings to be made at high data rates. The replay head amplifier was designed to preserve as much of the SNR of the replayed signal as possible before distribution to the rest of the signal conditioning circuitry. A Gated Cross-over detector was built to convert the replayed signal into a TTL compatible form suitable for input to the ***transputer***. The operation of the interface between the GXOs and the ***transputer*** preserved the event driven, message passing protocol, of ***occam*** channel communication, whilst converting between the parallel data format of the multiple-track recording head and the serial format of a ***transputer*** Link. Its event driven operation saved CPU time, and reduced the communication bandwidth of the acquisition process by a factor of approximately 9.5. A series of interface cards also allowed the ***transputer*** to control the compact-cassette tape transport mechanism, via the IBM PC.

Assembled together, the experimental apparatus created an environment where data could be generated and recorded using a multiple-track digital magnetic tape system, and subsequently replayed and analysed, all under the control of a ***transputer***. It also provided the environment in which programmes could be developed, and computer simulations performed.

2.6. References for Chapter 2.

DONNELLY, T., Time Delay Solenoid Switch using VMOS FETs., New Electronics, 4 February, 1986.

HOARE, C.A.R. Communicating Sequential Processes. Prentice-Hall, 1985.

IBM Corporation, IBM Personal Computer Hardware Reference Manual Library: Technical Reference. March 1986.

INMOS Ltd., Engineering Data Sheet: IMS T414 Transputer. August 1987, (a).

INMOS Ltd., Engineering Data Sheet: IMS C011 Link Adaptor. August 1987 (b).

INMOS Ltd., Transputer Reference Manual. Prentice Hall International (UK) Ltd., 1988 (a).

INMOS Ltd., Transputer Instruction Set: A Compiler Writer's Guide. Prentice Hall International (UK) Ltd., 1988 (b).

INMOS Ltd., Transputer Development System. Prentice Hall International (UK) Ltd., 1988, (c).

INTEL Corporation, Microprocessor and Peripheral Handbook, Volume II Peripheral, Santa Clara, USA, 1987.

JACKSON, T.J., Mapps, D.J., Ifeachor, E.C. & Donnelly, T. A Real-Time Transputer-Based System for a Digital Recording Data Channel.", Microprocessing and Microprogramming Vol. 25, pp 281-286, 1989.

MACKINTOSH, N.D. The Choice of a Recording Code, Int. Conf. Video and Data Recording, Southampton, I.E.R.E. Conf. Proc. No. 43, 1979.

SCANLON, L.J. IBM PC & XT Assembly Language: A Guide for Programmers. Brady Communication Company, Inc. NY, USA, 1985.

WHATTON, M.E. An economical Data Store. I.E.R.E. Conference Proceedings, No. 26, Video and Data Recording Conference, Birmingham, July 1973.

3. Theory, Modelling and Software of the Data Channels.

3.1. Overview.

In the course of the investigation, three data channels or systems were assembled, see figure 3.1. The core system, figure 3.1(a), was based on the compact-cassette hardware described in chapter 2, and is referred to as the compact-cassette system. The second, figure 3.1(b), was a model of a multiple-track tape system based on the compact-cassette system. The third system, figure 3.1(c), used elements from the previous two systems, combined with a digital waveform recorder, and was used to investigate a lateral head displacement compensation scheme. This chapter describes the software used to implement these systems, together with the relevant theory. All of the software presented in this chapter was written in ***occam***.

3.2. The Compact-Cassette System.

Figure 3.2 shows the operational flow-diagram of the compact-cassette system. The first step was to record the test data sequences onto the cassette tape. Pseudo-random Binary Sequences were generated in software and used as test sequence data. The system then operated in a semi real-time mode, the processor alternating between data capture and data processing. During the capture phase, the data and timing information describing up to $3x10^5$ transitions were stored. The capture process is then suspended and the data processed. This was repeated to the end of the test when the results and waveforms were filed onto the IBM PC's hard disc.

3.2.1. Generation and Encoding of Test Sequence Data.

The project was concerned with the recording channel, and not the information content of the data recorded. By using pseudo-random

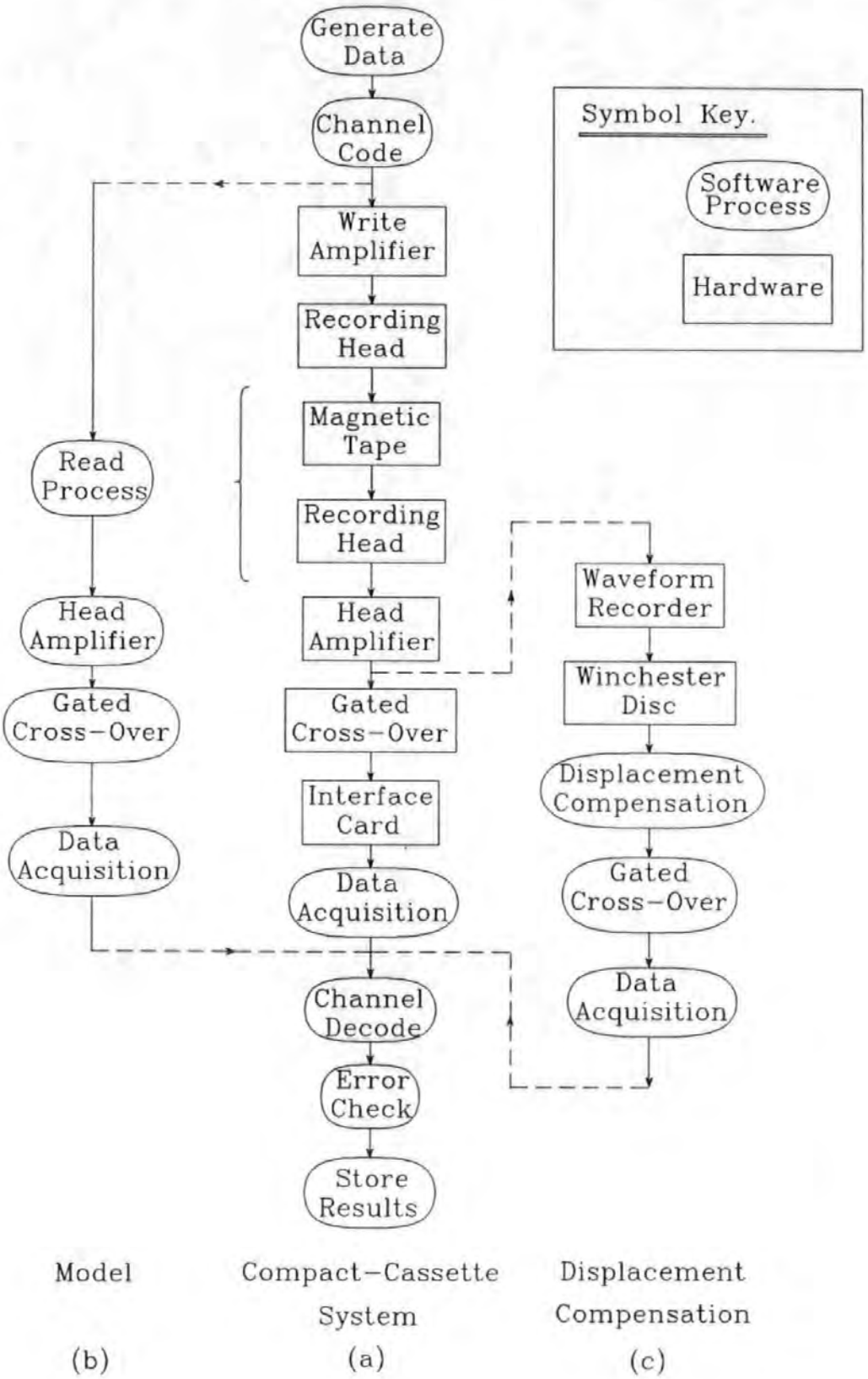


Fig. 3.1. The Three Data Channels used in the Investigation.

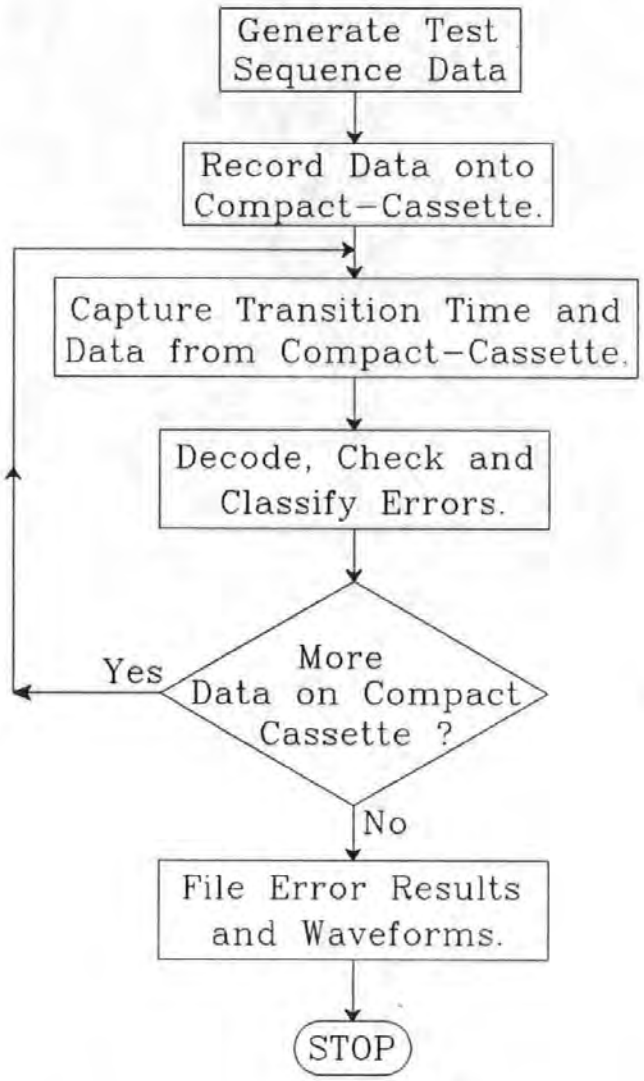


Fig. 3.2. Operation of the Compact-Cassette system.

sequences of data in the analysis, worst-case conditions were included in the investigation, ensuring any conclusions drawn were independent of information content and are therefore more widely applicable. Figure 3.3 shows the ***occam*** process map of the code used to generate and encode the test sequences for four tracks.

The channel modulation code chosen was Bi-Phase-L. This is a simple coding scheme that is widely used. It is not necessary to 'block-up' the data to be recorded, or use synchronisation words as it is simple to synchronise to, enabling a contiguous stream of data to be recorded.

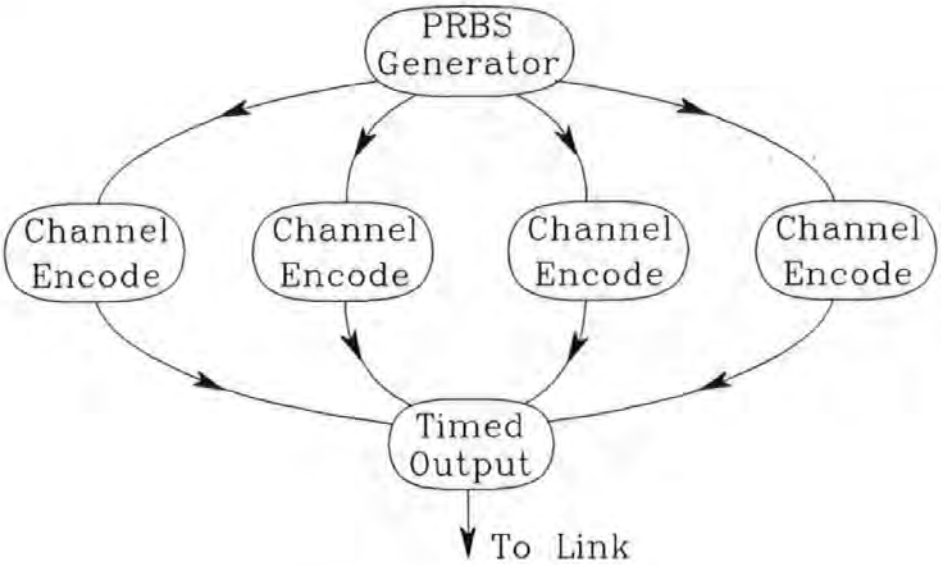


Fig. 3.3. ***occam*** Process Map for the Generation
and Encoding of the test sequences.

3.2.1.1. Pseudo-Random Binary Sequence Generator.

It is very difficult to generate truly random numbers sequences. However, it is straightforward for a computer to generate Pseudo Random number sequences. These are sequences that have a high degree of randomness, but are deterministically generated, are of fixed length and therefore repeat. A technique based on a Linear Feedback Shift Register (LFSR) (MacWilliams et al., 1976), a derivative of the Linear Congruential Method (Knuth, 1981, section 3.2.1) was chosen to generate the test data sequences.

The standard one dimensional LFSR was extended to two dimensions using an array (the array index providing one dimension, the word length the other), see figure 3.4. By using N bit words (rather than single bits), N PRBS sequences may be generated simultaneously for an N channel system. The LFSR was implemented as a circular buffer controlled by three pointers (FB1, FB2 and Inp in figure 3.4). The relationship between, and the length of sequences is determined by the initial state of the array and array pointers

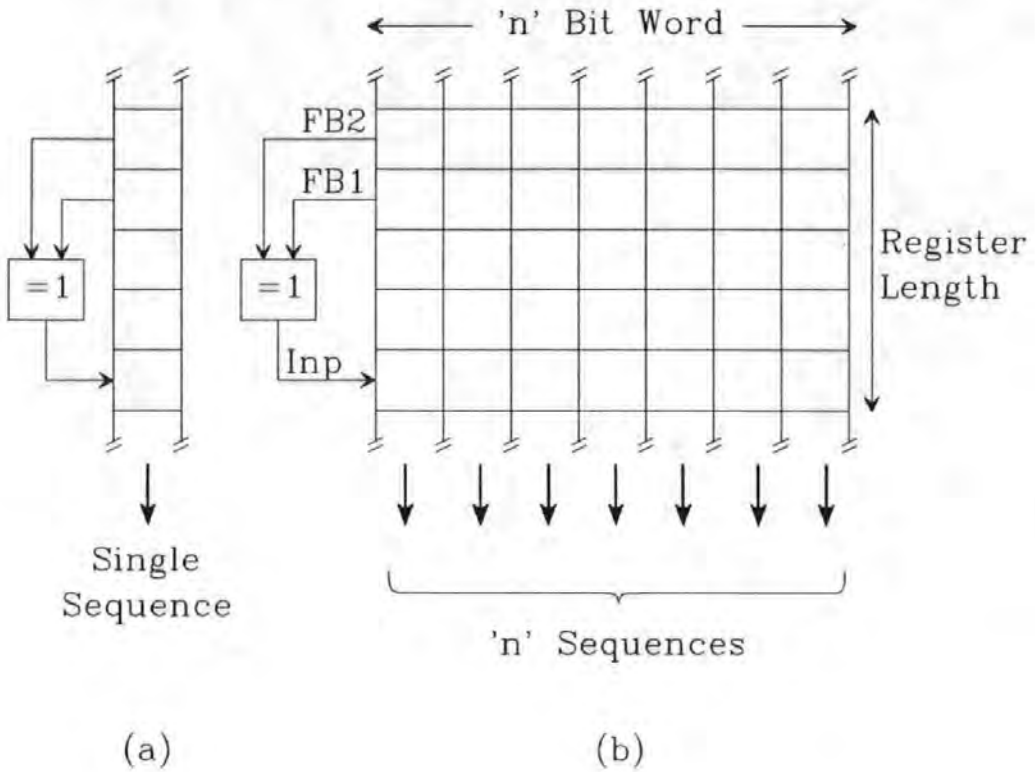


Fig. 3.4. (a) Standard one-dimensional LFSR (b) Generation of multiple Pseudo-Random Sequences.

respectively. All sequences generated were 'complete' or 'maximum length', and could be varied in length up to 32767 (2^{15}-1) bits.

3.2.1.2. Channel Encoding and Recording.

The channel code chosen was Bi-Phase-L (e.g. Mackintosh, 1979). It is very simple to implement in software: for each data bit received, two code bits are generated according to the following rules:

Data	Code Word
0	1 0
1	0 1

A separate timing process (run at High Priority to give a timing resolution of 1μS) output the channel encoded data at

timed intervals of 1/(2 * data rate) seconds. This is very simple to do in ***occam*** using the "delayed input" instruction. Although there is a latency of typically 19 processor clock cycles before the process is scheduled, this latency is essentially constant for a lightly loaded ***transputer*** (as was the case during recording), and can therefore be ignored. The data are output to the write amplifier for recording, via one of the ***transputer***'s Links.

3.2.2. Decoding and Analysis of Replayed Data.

Figure 3.5 shows the ***occam*** process map used to decode and analyse the data for the four-track system. The compact-cassette system operated in a semi real-time mode: real-time capture of the data, followed by off-line processing. New data were read from the hardware

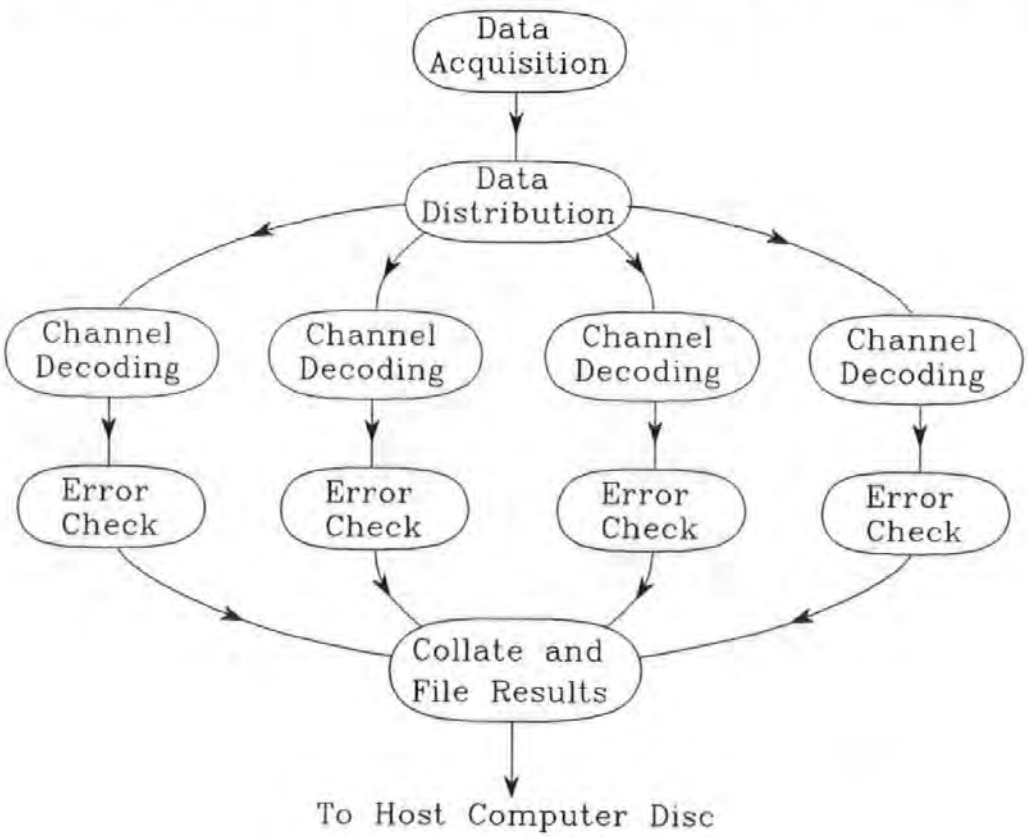


Fig. 3.5. ***occam*** Process Map for the Decoding and Analysis of Replayed Data.

via one of the *transputer*'s Links. The data acquisition process also recorded the time the data were read, storing the data in one array, and the time in a second array. When the arrays were full, the data aquisition process was suspended whilst the data were processed. The data were demultiplexed with respect to their track, and processed separately on a track by track basis.

3.2.2.1. Data Acquisition.

This section is used to illustrate several fundamental aspects of the *occam* programming language and *transputer* introduced in section 2.2.1, and is therefore considerably more detailed than similar sections.

The hardware interface between the GXO detectors and the *transputer* was Event or Data Driven. When new data were detected the current state of all the channels was made available for the *transputer* to input via a Link. As the operation of this interface was designed to conform to *occam* channel protocol, the code to read this data is simply:

```
link ? new.data
```

When the *transputer* executes this line of code, it checks to see if there is a message waiting to be read from channel "link". If there is then it is stored in the variable "new.data" and the process continues being executed.

If no message is waiting (i.e. no transitions have occurred) the process is suspended and put on the inactive process list. When the message does arrive (i.e. a transition has occurred) the process is moved to the active process list. When the process comes to the top of this list, it is rescheduled. The message transfer can then occur (i.e. the message is stored in "new.data") and the process continues being executed.

In order to perform the decoding process, the time the transition occurred was required. This was performed by extending the

above code to:

```
SEQ
  link ? new.data
  clock ? transition.time
```

This is read as "SEQentially, read new data from channel "link", then read transition time from "clock". The simplistic elegance of these three lines of *occam* demonstrates its applicability to real-time applications. However, what is not immediately obvious from this code are the time intervals that may exist between the transition occurring and the data being read and between the data being read and the reading of the clock.

If a transition occurred immediately after the process was descheduled, the transition would not get read until the process had reached the top of the active process list and been rescheduled. The length of time this takes is dependent on the number of other processes preceding it on the list. This time interval is therefore variable and potentially unacceptably long.

Of equal undesirability would be for the process to come to the end of its 'time-slice' period immediately after reading the channel. It would then get descheduled before the time was recorded. When eventually rescheduled, the time recorded may be considerably different to the time the channel was read (or indeed when the transition occurred).

In an attempt to remedy this type of problem, *occam* allows Prioritization of processes (INMOS, 1988(b), Welch, 1987, Burns et al., 1987). Low priority processes only get scheduled if no High priority process can proceed. Running the data acquisition process at High priority results in it being rescheduled immediately the channel communication can proceed, and prevents it being descheduled before the time has been recorded. A transition can therefore be timed to a determinable accuracy (calculated later).

If n = number of tracks and r = data rate, the average time available to process each data bit is,

$$t_{proc} = {}^{1}/_{(n * r)} \text{ seconds}$$

However, as Bi-Phase-L has 1.5 transitions per data bit on average, in a worst case situation as many as 1.5n transitions may occur in t_{proc}. One solution is to use a data buffer process between the data acquisition process and the data processing (Pountain, 1988). This decouples the two processes, allowing data to be captured in short bursts at high rates, and processed at the average rate.

Putting the above code into a loop, and adding a fifth line that outputs the transition and its detected time via the channel "to.buffer", results in:

```
WHILE TRUE
SEQ
  link ? new.data
  clock ? transition.time
  to.buffer ! new.data ; transition.time
```

Consider the worst case situation of a sequence of transitions, closely spaced, yet non-coincident. The sequence of events would be:

i) The first transition occurs and the data becomes valid to be read from the Link.

ii) The acquisition process is scheduled, the data read, the time recorded, and the message output on channel "to.buffer" initiated.

iii) The acquisition process is descheduled as the channel communication cannot complete (until the data is received by the buffer process).

iv) The buffer process is scheduled, and the data and transition time read from the channel.

v) The Channel communication completes. The acquisition process can now proceed and so the buffer is descheduled (being a lower priority) to allow this.

vi) Steps i), ii) and iii) are repeated. However, the second message communication to the buffer cannot complete until the buffer has finished processing the first communication (it was descheduled immediately after its receipt). This results in the High priority acquisition process waiting for the Low priority buffer process; precisely what was not wanted.

The standard technique to deal with this problem is to run the buffer process at High priority too. This results in the buffer process not being descheduled until it has stored the first message. However, it also results in the time taken to store the data in the buffer process being added to the data acquisition time, degrading timing resolution.

Figure 3.6 shows a further optimization of this technique where the acquisition and buffer processes, implemented as a circular buffer (Burns, 1988), has been combined to reduce the time involved in scheduling processes.

```
WHILE TRUE                                              4
  PRI ALT                                               74
    from.link ? tran.data.buffer[in.pointer]            28
      SEQ
        clock ? tran.time.buffer[in.pointer]            11
        in.pointer := (in.pointer PLUS 1)               6
        IF
          (in.pointer = buffer.size)                    8
            in.pointer := 0                             2
          TRUE
            SKIP
        buffered.items := (buffered.items PLUS 1)       6
    (items.in.buffer>0) & data.request ? message        24
      SEQ
        data.out ! tran.data.buffer[out.pointer] ;
            tran.time.buffer[out.pointer]               72
        out.pointer := (out.pointer PLUS 1)             6
        IF
          (out.pointer = buffer.size)                   8
            out.pointer := 0                            2
          TRUE
            SKIP
        buffered.items := (buffered.items MINUS 1)      6
```

Fig. 3.6. Data Acquisition and Circular Buffer.

The figures to the right of the code indicate approximate execution times (in CPU cycles). It therefore takes 4+74+61=139 cycles to input and time a transition, and 4+74+118=196 cycles to output the data and the time it occurred. For a 15MHz T414 *transputer* with a 67nS CPU cycle time, the maximum burst data acquisition rate is therefore 1/(139 * 67nS)=108k transitions per second (where the maximum burst length corresponds to the buffer

size), and the maximum continuous data acquisition rate is 1/((139+196)*67nS))=45k transitions per second. This second figure also indicates the worst-case timing resolution of 1/45k=22μS.

The maximum continuous data rate stated above assumes the acquisition process to be the only process running on the *transputer*. This was not the case. The decoding, error checking and classification processes for all the channels were also run concurrently on the same *transputer*. The performance degradation caused by these other processes resulted in the single *transputer* being unable to process the data from the four tracks at the required rate of 5kbps per track in real-time, hence the pseudo real-time operation of the compact-cassette system.

There is a second factor that can degrade performance. The figures quoted above for the number of CPU cycles assumes both programme and associated data reside in the fast internal memory of the *transputer*. External memory references are slower, and may therefore degrade performance. For the *transputer* board used, each external memory reference required an extra 3 processor cycles. However, the *transputer*'s 4 byte instruction pre-fetch removes this delay for linear code sequences. After a branching operation or process-swap the pre-fetch does not help, and the programme execution will be delayed whilst the memory reference is made.

The severity of the degradation is therefore dependent on the code sequence. From simulation (INMOS, 1988), performance timings should be extended by approximately 60% when the programme and data reside in external memory. To maximise performance (Atkin, 1987), it is possible to arrange for specific processes to reside in the *transputer*'s internal memory. The code for the data acquisition process (being the most performance critical) was therefore placed in internal memory. However as the internal memory of the T414 *transputer* is only 2K bytes in size, the majority of the programme and data resides in external memory, and was therefore subject to this degradation (although CPU cycles quoted do not reflect this).

As it was not possible to run the code in real-time on a single *transputer*, a pseudo real-time mode was used. The data

acquisition process was optimised to take full advantage of this, and is shown in figure 3.7.

```
SEQ index = 0 FOR array.size                    13
  SEQ
    from.link ? transition.data.array[index]    28
    clock ? transition.time.array[index]        11

                                          Total 52
```

Fig. 3.7. Code for Pseudo Real-Time Data Acquisition and Buffering.

The maximum burst data acquisition rate is now the same as the continuous rate at 1/(52*67nS)=288k transitions per second, with a timing resolution of approximately 3.5μS. Therefore, operating in a pseudo real-time mode not only removes the constraint of having a limited amount of processing time, it also considerably improves the maximum data acquisition rate and timing resolution. The only penalty is a limit on the maximum number of contiguous transitions recorded, but as this was approximately $3x10^5$ this was not considered too limiting.

Figure 3.8 illustrates how the track transition data and timing information (for just three tracks for simplicity) is stored by the data acquisition process. Due to a one bit buffer in the hardware, the time of the transition marking the start of the data item stored in location n of the data array, is stored in location n-1 of the time array (for clarity, this has been purposely overlooked in the description).

3.2.2.2. Distribution of Data for Concurrent Evaluation.

At the end of the data acquisition phase, the data for all the tracks were stored in one array, with their respective transition timing information in a second array. The first stage in the processing of this data was to separate the data associated with each track and distribute it for processing on a track-by-track basis. This is shown

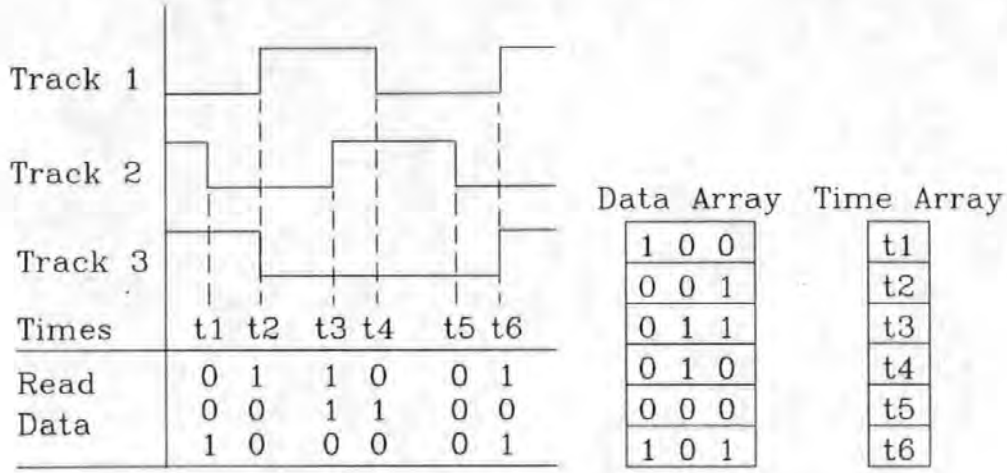


Fig. 3.8. Multiplexing the Data and Timing Information for three tracks into two arrays.

in figure 3.9. For example, track 1, the data changed to a 1 at time t2, to a 0 at time t4 and to a 1 at time t6. This was sufficient information to perform the channel decoding.

The transition times were stored as 32 bit words, whilst the data occupied a single bit in a byte. It would have been inefficient to output the single bit on its own and therefore the data bit was combined with the time word. The time word was shifted by one bit to the left, losing the MSB, and the data bit was introduced at the now empty LSB position, as illustrated in figure 3.10.

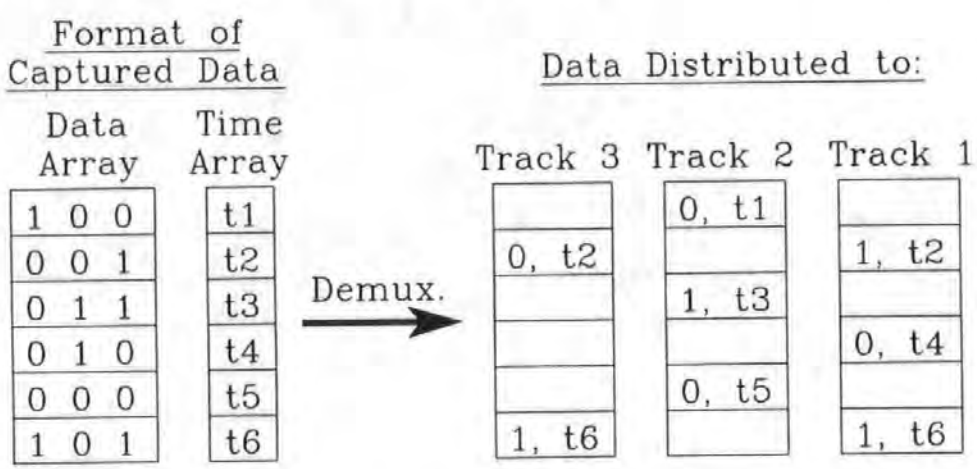


Fig. 3.9. Demultiplexing the Timing and Data Information with respect to Track.

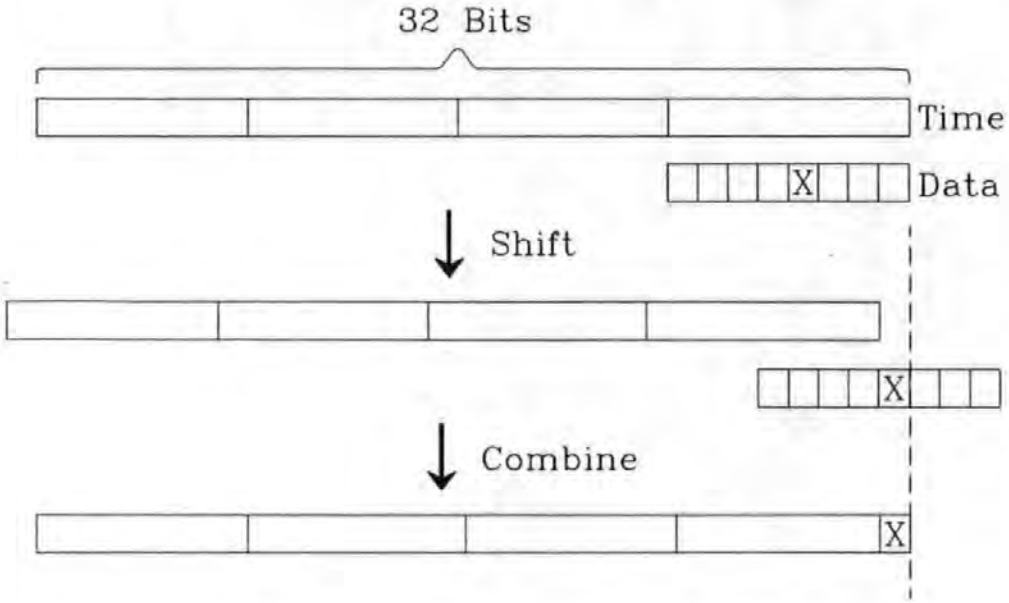


Fig. 3.10. Combining the Timing and Data Information.

If the data bit had replaced the LSB of the time word, the timing resolution would be halved. This would be undesirable. Losing the MSB of the time word halves the maximum measurable period (to approximately 2000 seconds), which is of little consequence.

'Time-stamping' data in this way decouples it from other time dependencies, simplifying further processing (Jackson et al., 1989). This is especially beneficial in multi-processor systems as it removes the need for time synchronisation algorithms (Carlini et al., 1988).

3.2.2.3. Bi-Phase-L Channel Decoding.

The first stage in the decoding process was synchronisation i.e. determining whether transitions occurred at data bit centres or boundaries. The use of Bi-Phase-L considerably eases this task compared to most channel codes: whenever the period between two transitions equals (within some tolerance) twice the code bit period, the second transition occurred at the data bit cell centre, see figure 3.11.

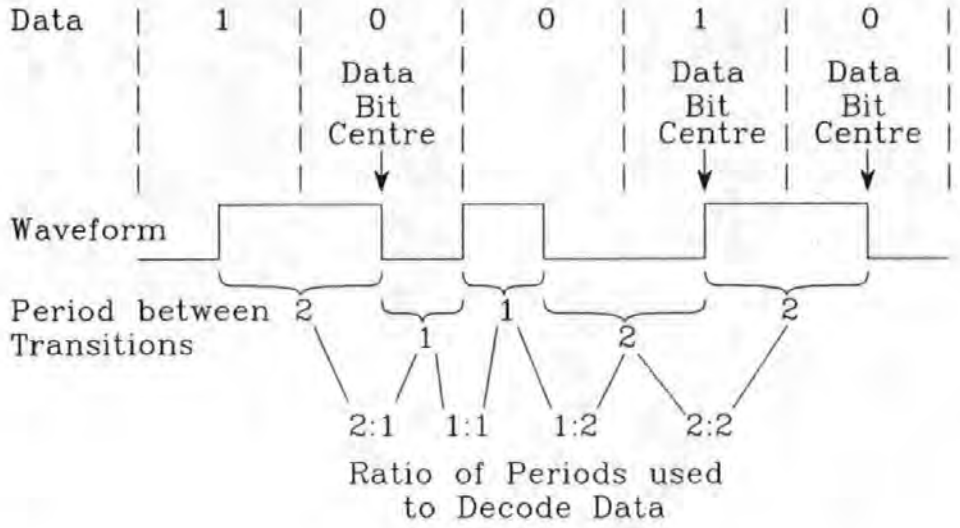


Fig. 3.11. The use of Timing Ratios to Synchronise and Decode Bi-Phase-L Sequences.

Once synchronised, the decoding process used the same 'ratio of periods' to determine sample points, and from this, decoded the sequence. The decoding process, detailed in figure 3.12, is divided into two parts; i) detection of valid inter-transition periods, and ii) a 'free-wheeling' mode to cope with missing data. If a period greater than the data bit period is detected, data have been lost. The 'WHILE' loop in figure 3.12 outputs the same number of data bits that were lost, maintaining the time-base. This performs a function similar to a Phase-Locked-Loop in a hardware decoder.

3.2.2.4. Error Detection, Classification and Logging.

Errors were detected by regenerating the PRBS and comparing it with the received data. This was performed on a bit-by-bit basis: as each bit was received, the PRBS was clocked to produce the next bit in the reference sequence for comparison.

Before any comparisons were made, the reference PRBS was synchronised with the incoming data. It is a feature of sequences produced by Linear Congruential Shift Registers that for an n stage

```
Synchronisation (find period 2 code bits wide)
REPEAT
  Get Time t3 and Data d3 of next transition
  Calculate period from last transition, p2 = (t3 - t2)
  Convert period to multiples of the code bit period
  IF
    1:2 or 2:2
      past sample-point, output data (d3)
      calculate next sample-point
    2:1
      calculate next sample-point
    1:1 AND past sample-point
      output data (d3)
      calculate next sample-point
    ELSE -- invalid period sequence detected
      WHILE sample-point not reached
        output data (d3)
        increment sample-point by data bit period
UNTIL End Of Data
```

Fig. 3.12. Pseudo-code Of The Channel Decoding Process.

register, the i^{th} value of the sequence can be calculated from the previous n values. Therefore simply filling the reference PRBS register with the incoming data automatically synchronised the two data sequences (assuming the register was filled with correct bits).

Rather than simply count individual bits in error, a more detailed error classification scheme was devised. Initially, the use of a multi-dimensional array as a look-up table was investigated. The detected periods formed the indexes into the array, the contents of which specified the decoded data sequence or the error sequence. Although successfully implemented for the data sequence 0011 (for which there are 64 possible period combinations that had to be calculated), a general purpose algorithm was developed that was independent of code sequence and length.

Figure 3.13 shows three error sequences. In sequence (a) two bits are in error. Rather than simply record this as two single-bit errors, it was also recorded as a double-bit error. Sequence (b) also appears to contain two single-bit errors. However, this may have been caused by a three bit wide drop-out. During a multiple-bit drop-out, statistically, 50% of the received data will match the reference data (on average). Sequence (b) was therefore classified as a triple-bit

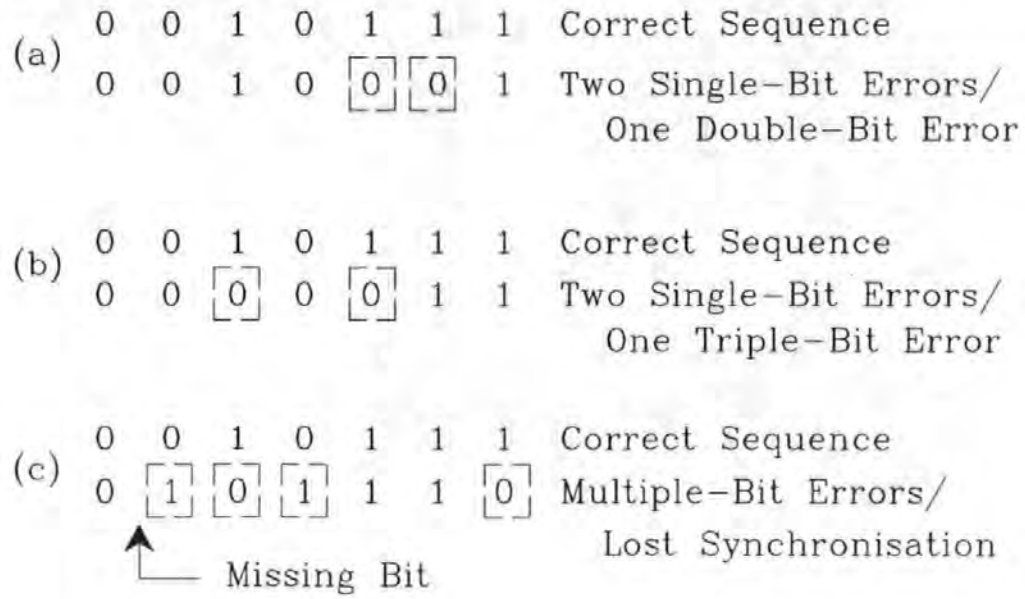


Fig. 3.13. Classification of Errors.

error as well as two single-bit errors.

Sequence (c) appears to contain 4 closely spaced single-bit errors. These errors were caused by a single missing bit. If the reference and received data sequences became unsynchronised, statistically, 50% of all succeeding received bits would be classified as in error. Therefore a 'Maximum Burst Error Length' was specified. If this maximum is exceeded, a single 'Lost Synchronisation' error is recorded, and the two sequences are resynchronised. The following error counts were maintained during testing:

Individual Correct Bits.
Individual Bits in Error.
Bits Classified as being Correct.
Bits Classified as being in Error.
Errors of Burst Length 1, 2,...'Maximum Burst Error Length'.
Number of times Synchronisation was lost.

Figure 3.14 shows the flow-chart of the algorithm used to classify errors. A consequence of this classification strategy was that many intermediate results needed to be stored during burst

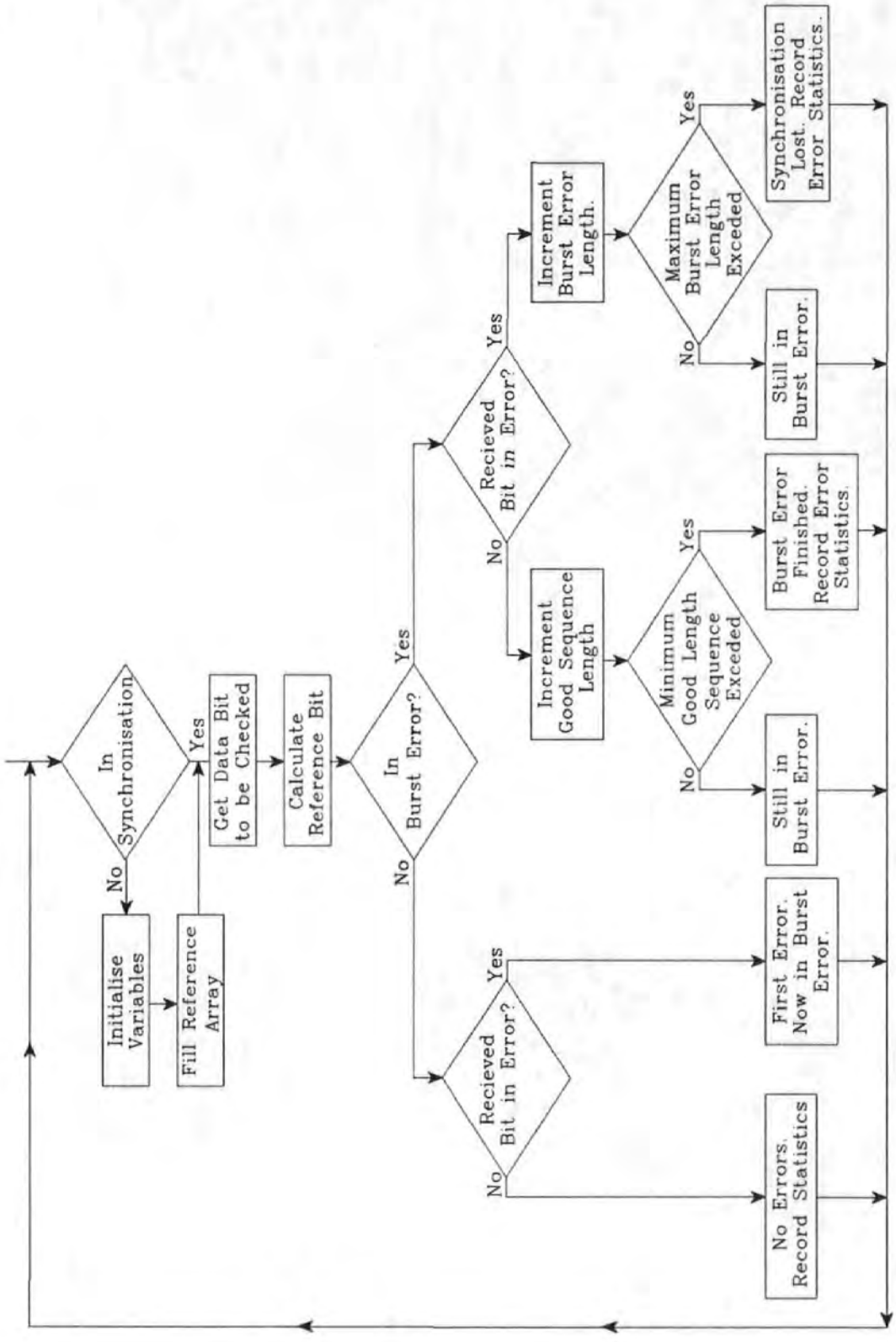


Fig. 3.14. Error Classification Flow Chart.

errors, as the final result was not known until either 'minimum good' bits or 'maximum bad' bits had been received.

This classification scheme gives a more accurate measure of the systems performance than does the raw bit error rate (i.e. the total bits in error divided by the total number of bits received. Random data achieves a raw error rate of 0.5, whilst it would be classified at nearer zero (as it intuitively should) using this scheme.

When all the data stored during the acquisition phase were checked, the error counts were added to the total error counts for the test. When the last block of data had been captured and processed, the total error counts for the test were filed on the host computer's hard disc.

3.3. The Model of the Compact-Cassette System.

Figure 3.15 shows the operational flow-diagram for the model of the compact-cassette system. The model operated in a similar manner to that of the compact-cassette system, alternating between data capture and data processing. However, the operating cycle was extended to allow changes to be made to various model parameters. The following set of model parameters were read from a text file for each block of data to be simulated:

Simulation Data Rate.
Read Track Width.
Write Track Width.
Side-Write Width.
Threshold level of Polarity Discrimination Comparator.
Threshold level of Peak Centre Comparator.
Track Displacement
Amount of Data Skew between Tracks.
Maximum number of Errors before assuming synchronisation lost.
Minimum number of correct bits between Burst Errors.
PRBS Register Length.

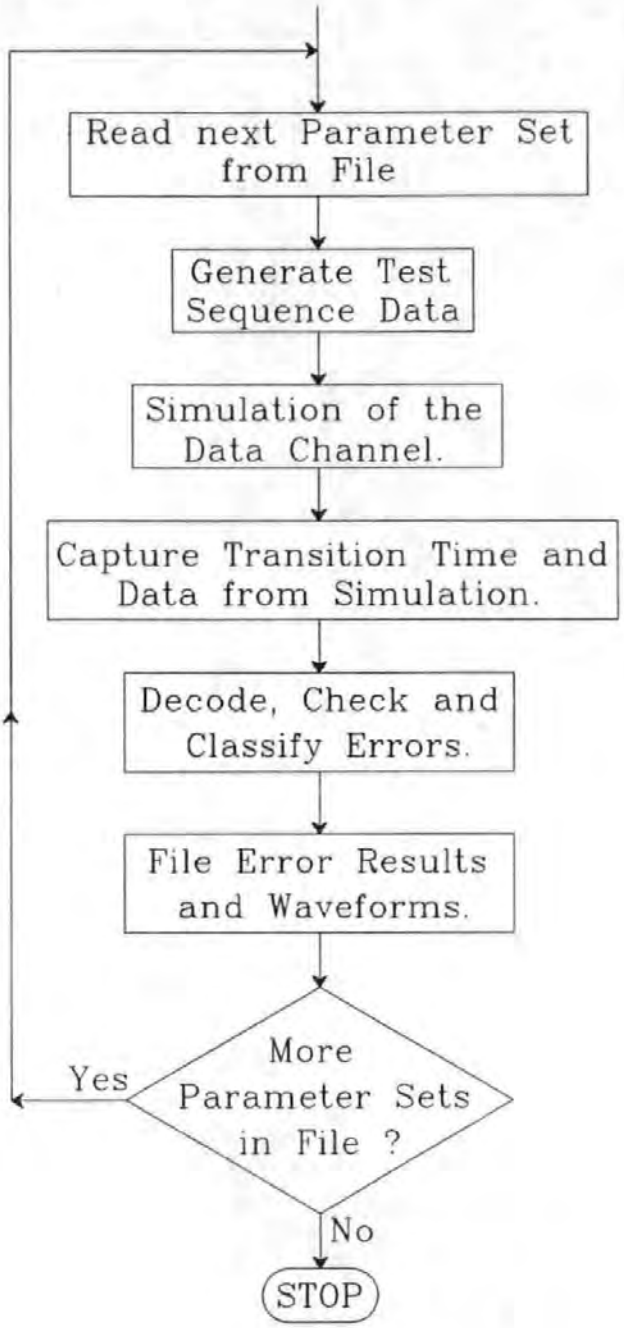


Fig. 3.15. Operation of the Model.

Multiple sets of parameters may be stored consecutively in the same parameter file, enabling whole sequences of tests to be initiated one after the other without intervention.

There was also the option to record not only the error results for a particular test, but also a 'snapshot' of the simulated

waveforms. For each track the following sampled data waveforms could be recorded and stored on the IBM PC's hard disc:

- Head Amplifier, input and output (analogue).
- Gating signal, comparator input (analogue) and output (digital).
- Polarity signal, comparator input (analogue) and output (digital).
- GXO detector output (digital).

It would take 14K bytes of memory to store the complete waveform for one 1kbps 7-bit PRBS, at a sampling frequency of 500kHz, therefore the waveform data was decimated in time to reduce memory usage. The decimation factor was calculated at run-time to make maximum use of the memory declared for waveform storage (normally 500 words per waveform).

As neither the host computer, nor the TDS, provided support for graphical display, a separate display package, TellaGraf (TellaGraf, 1987), running on the Polytechnic's central PRIME minicomputer was used for display and plotting of waveforms. The raw waveform data were encapsulated with the relevant TellaGraf commands to produce a correctly scaled, annotated and positioned display. The waveform data plus commands form a TellaGraf Command File, which may be directly input to the package. Figure 3.16 shows a typical set of waveforms from the model. This display facility was an invaluable feature of the model; during system development as well as during simulated investigations.

The data generation and channel encoding, as well as the decoding, error checking, classification and storage of results used in the model was the same as that used in the compact-cassette system. Being able to use exactly the same code in the two systems not only saved time, more importantly it guaranteed consistency between the two systems. The following sections describe the processes used to model the compact-cassette system.

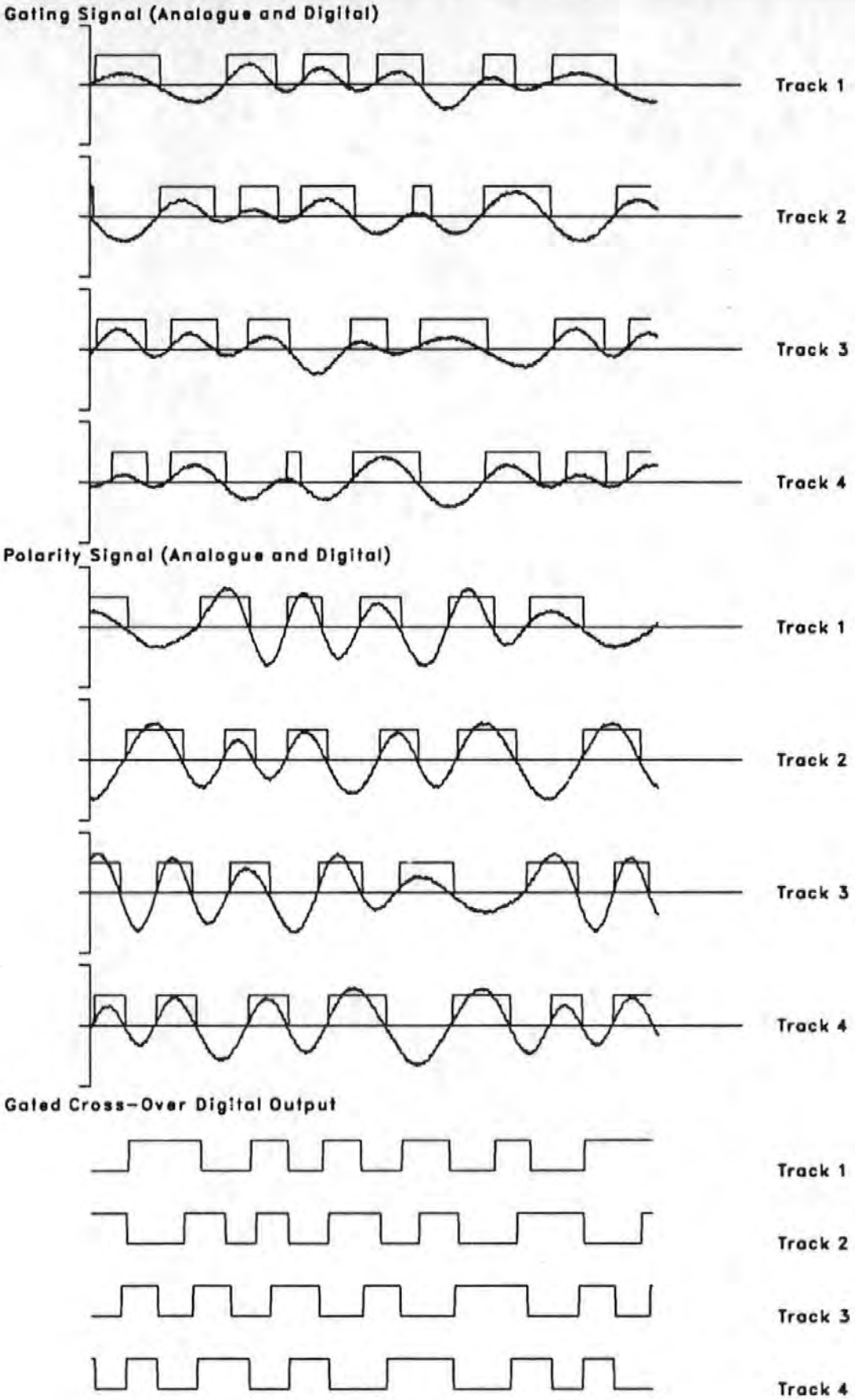


Fig. 3.16. Typical Waveform Output from the Model.

3.3.1. Generation of Gaussian White Noise.

Several sections of the model required a Gaussian or Normally distributed noise source. In common with most computer systems, the TDS has a standard random number generator. Although this produces a Uniformly distributed pseudo-random number sequence, a Gaussian distribution may be derived from it.

The most popular method of generating pseudo-random number sequences is the Linear Congruential Method, defined by,

$$x_{n+1}=(ax_n+c) \text{ MOD } m \qquad \text{Equ. 3.1}$$

where x_{n+1} is the next number in the sequence. The length and degree of randomness of the sequences generated is determined by the values of a, c, m and x_0, and must therefore be chosen with care. The TDS uses this algorithm with:

$$a=1664525 \qquad c=1 \qquad m=2^{31} \qquad x_0<>0$$

Several methods for generating pseudo-random sequences with a Gaussian distribution exist (Knuth, 1981, section 3.4.1). However, the simpler methods take a long time to compute, whilst the fast ones are complex to implement. An algorithm that produced an approximation to the Gaussian distribution was therefore used. The scaled summation of several uniformly distributed number sequences can be used to produce an approximation to a Standard Normally distributed sequence (Gordon, 1978). If x_i is the i^{th} number in a uniformly distributed sequence, then the number sequence defined by,

$$z_n=\frac{\sum_{i=1}^{k} x_{nk+i} - (k/2)}{(k/12)^{0.5}} \qquad \text{Equ. 3.2}$$

approximates to a Standard Normal distribution. This may be

transformed into a Gaussian distribution using,

$$x = z\sigma + \mu \qquad \text{Equ. 3.3}$$

where σ is the Standard Deviation and μ is the Mean. The approximation improves as k increases. A convenient value of k that gives a good approximation is 12. This reduces the summation to:

$$z_n = \sum_{i=1}^{k} x_{nk+i} - 6 \qquad \text{Equ. 3.4}$$

A procedure was written that generated Gaussian White Noise (GWN) ('White' because it contains all frequencies) using equation 3.4 above, where values of x were generated using the TDS function RAN (INMOS, 1988a).

3.3.2. Model of the Replay Channel.

A Sampled Data model with a sample period of 2μs, was used to model the read process and analogue electronics of the compact-cassette system. This may seem an unnecessarily high sampling frequency (500kHz) to simulate a 5kHz waveform. However, 2μs is the nearest convenient sampling frequency to the 3.5μs timing resolution of the compact-cassette system, allowing direct comparisons to be made.

It is important to note, a general purpose simulation programme was not written. General purpose solutions to problems such as deadlock, race-conditions e.t.c. (e.g. Dowsing, 1985, Nevison, 1989, Djahanguir et al., 1989) did not therefore need to be solved. This greatly simplified the design of the model and enhanced performance. Solutions to the above problems were dealt with individually and wherever possible internally to the modules, maintaining a modular approach that simplifies future modifications.

The first stage in the simulation was the generation of the

analogue waveform representing the signal from the replay head. Linear Pulse Superposition was used to generate these waveforms. The shape and width of the isolated pulses have a large bearing on the accuracy of the model, and were therefore carefully derived. Once the basic waveform had been generated, various error sources were introduced. These included drop-outs, amplitude fluctuations, lateral head displacement and noise.

Digital filters were used to model the components of the signal conditioning circuitry, i.e. the Head Amplifier and Gated Cross-Over detector. All sources of electronic noise were lumped together and incorporated at the analogue to digital conversion stage. The digital output from the simulated Gated Cross-Over detector was 'time-stamped' and stored in an array, in exactly the same manner as the compact-cassette system, for decoding and analysis. The ***occam*** process map for this section of the model is shown in figure 3.17.

3.3.2.1. Linear Pulse Superposition.

Figure 3.18 shows an isolated flux reversal or transition together with the single isolated replay voltage 'pulse'it generates. Linear Pulse Superposition (LPS) (Mallinson et al., 1969) applied to magnetic recording states that the voltage waveform produced by a series of flux reversals is the algebraic sum of a series of isolated pulses, centred on the flux reversals.

Expressed mathematically, the combination of n isolated pulses, f(t), separated by T/2, where T = 1/data rate is given by,

$$e_{resultant}(t) = \sum_{n} (-1)^n . f(t + n\,T/2) \qquad \text{Equ. 3.5}$$

Once the shape of the isolated pulse has been determined, the replay voltage waveform for any recorded data sequence, at any packing density, may be generated by combining pulses with the appropriate spacings.

3.3.2.2. Determination of Isolated Pulse Shape.

The accuracy of simulation of a waveform using LPS is dependent on the shape and width of the isolated pulse used. Equation 3.6 (developed in appendix B) states the voltage produced by a single magnetic transition (of arctangent form):

$$e_x(X) = C_3.\ln\left[\frac{(f_x+d+\delta)^2+X^2}{(f_x+d)^2+X^2}\right] \qquad \text{Equ. 3.6}$$

where C_3 = constant of proportionality (given in Appendix B)
f_x = arctangent parameter
d = head to tape spacing
δ = Medium thickness

A further simplification may be made by assuming (near) zero medium thickness,

$$e_x(X) \propto \frac{1}{1 + (x / (d + f))^2} \qquad \text{Equ. 3.7}$$

This is referred to as the Lorentzian shape pulse. Several methods exist to experimentally estimate the parameters in this expression (Loze et al., 1990). However, the objective was to simulate a specific recording channel. The mathematical function used to generate the isolated pulse does not need a rigid theoretical basis. It is only necessary for the isolated pulse generated to match that of the system to be simulated.

Mackintosh (Mackintosh, 1979(b)) investigated nine analytical expressions, including the Lorentzian, concluding that:

$$f(x) = \frac{1}{(1+x^2+x^4)} \qquad \text{Equ. 3.8}$$

produced the isolated pulse most representative of the systems under investigation. The shape of the isolated pulse this generates is

shown in figure 3.19, together with that from the inductive head. It can be seen that this is not an accurate fit, as were none of the other expressions Mackintosh investigated and dismissed.

The pulse used by Mackintosh was symmetrical, whereas the pulse from the compact cassette system was asymmetrical. This asymmetry was attributed to the perpendicular or y component of the magnetic medium, and would be zero in an ideal longitudinal medium. A closer match was found using different expressions for the left and right side (as suggested by Mackintosh), but significant differences were still apparent, especially at the base of the pulse. It was therefore decided to find new analytical expressions for the pulses.

The first step was to determine the shape of the isolated pulses from the heads to be modelled. Sixteen such pulses (four per track) were captured at random using an oscilloscope, and subsequently plotted onto paper. For each pulse, twelve time periods were measured, referenced to the pulse peak, see figure 3.20.

The sixteen values for each time period were averaged to produce a reference pulse. It was suggested (Good, 1989) that an analytical expression for the reference pulse could be determined using standard numerical analysis routines. This task was considerably more involved than initially thought. A single analytical expression that accurately described the complete pulse could not be found. The main problem was discontinuities in the waveforms. Although many equations could be found that gave a good overall fit, they all contained at least one discontinuity. Figure 3.21 shows two such curves, each with a discontinuity. However, the pulses could be accurately modelled by carefully combining two or three expressions that had their discontinuities in different parts of the curve. Each expression had the form:

$$f(t) = \frac{a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4}{b_0 + b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4} \qquad \text{Equ. 3.9}$$

A small FORTRAN-77 programme (listed in Appendix C) was written to determine the coefficients. This task was viewed as the solution of a set of simultaneous linear equations. The NAG routine

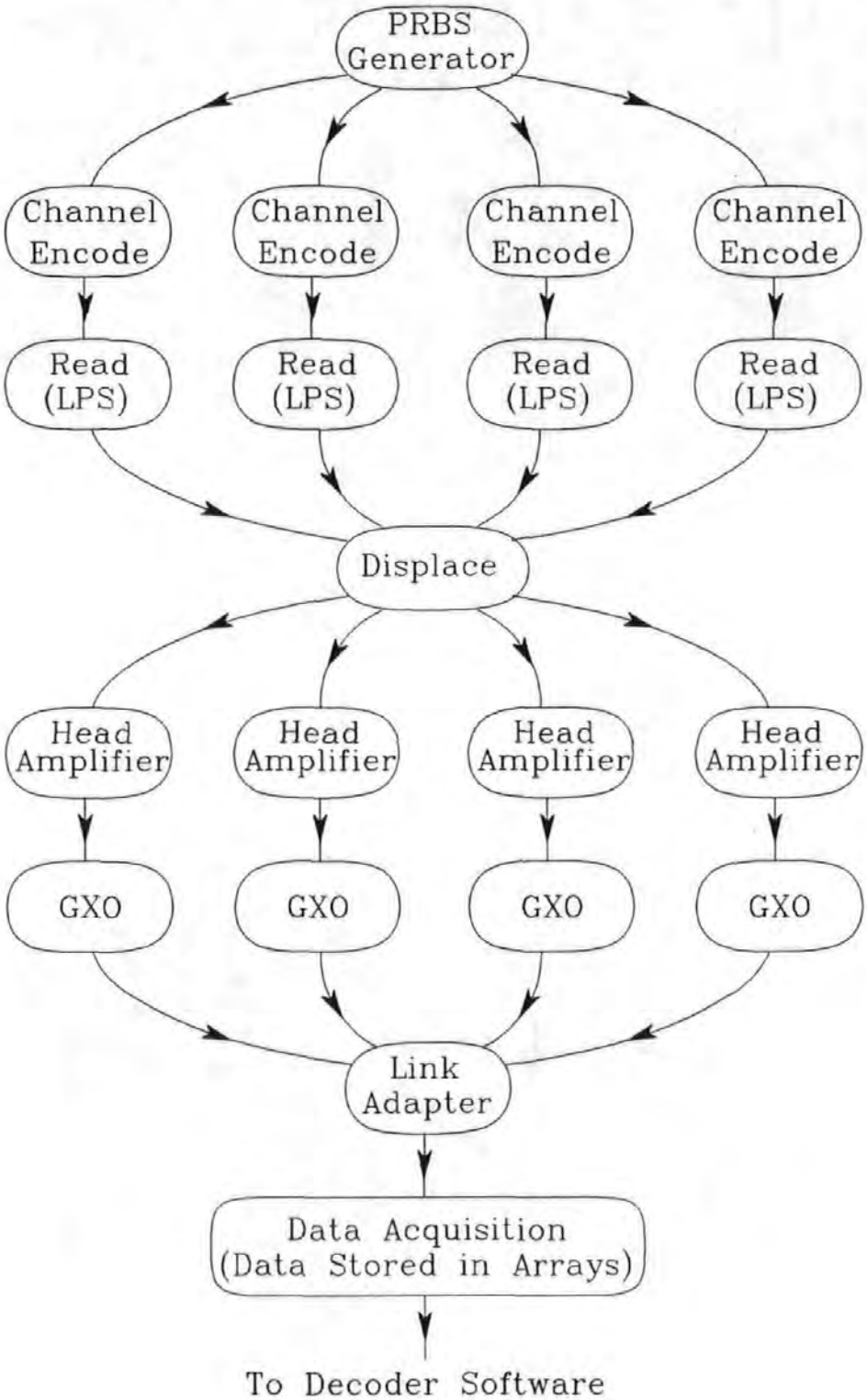


Fig. 3.17. ***occam*** Process Map for the Replay Channel Model.

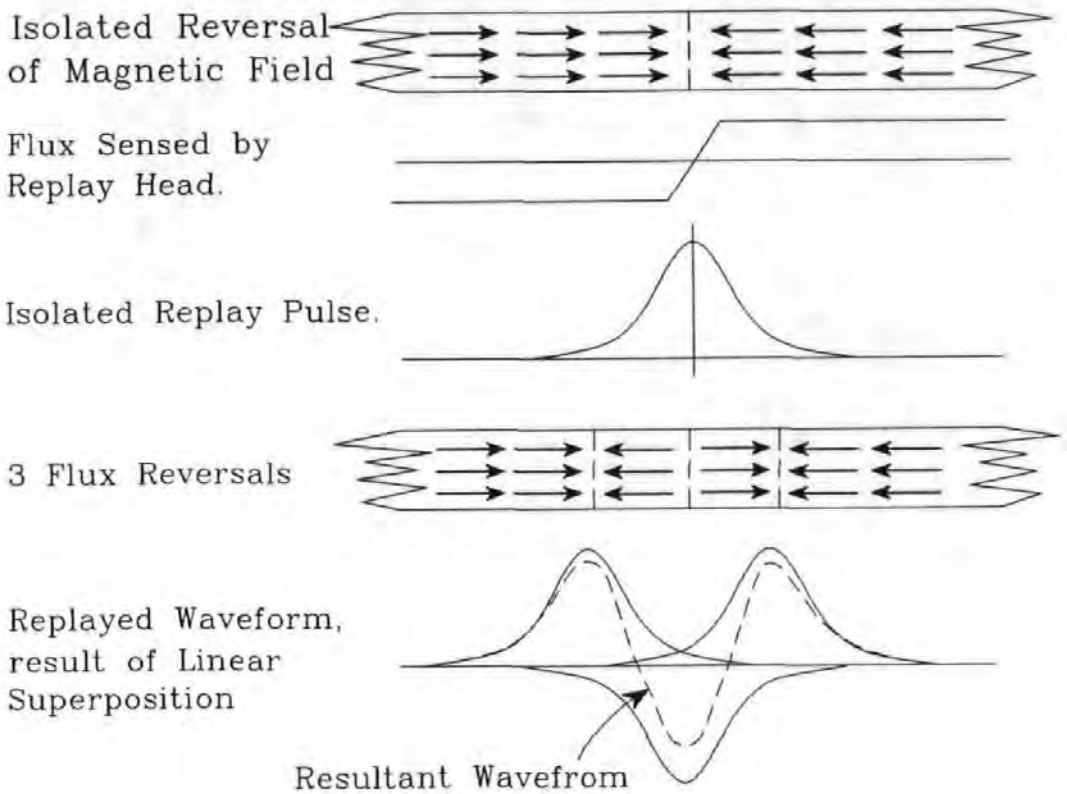


Fig. 3.18. Linear Superposition applied to Magnetic Recording.

As the data rate increases the isolated pulses overlap and interfere with one another more. This does not cause a problem when they are regularly spaced: the interference is regular and symmetrical. Irregularly spaced pulses - as is the norm - combine to produce a waveform where the peaks vary in height and position. At sufficiently high data rates the amount of peak shift and peak attenuation due to this Inter-Symbol Interference (ISI) can cause errors. For each code there is a specific combination of pulse spacings that will cause the largest peak shift and attenuation. This worst-case combination is normally a sequence of pulses spaced by the smallest amount the code allows, immediately followed by a sequence of pulses spaced by the largest amount the code allows.

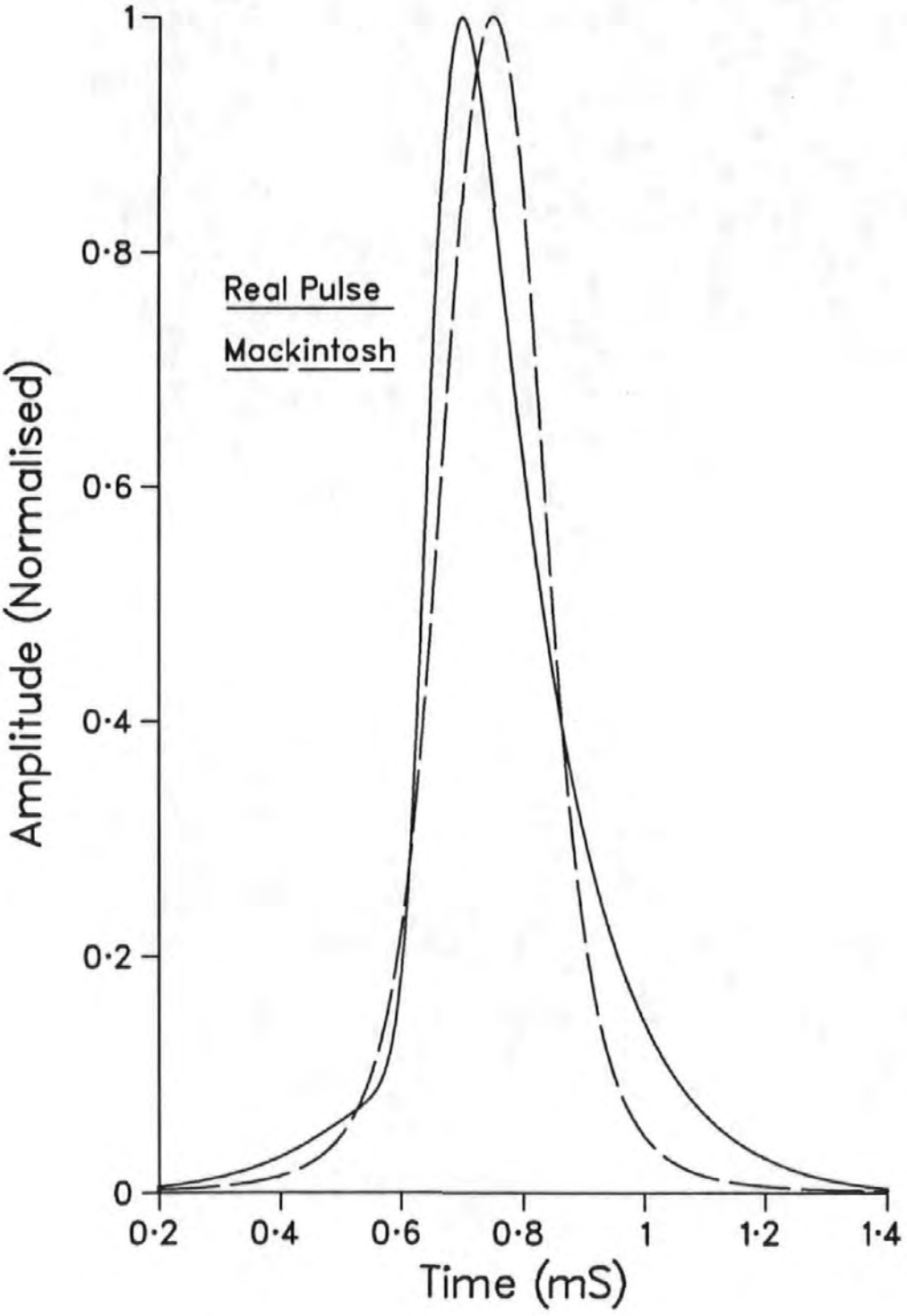


Fig. 3.19. Isolated Pulse Shapes.

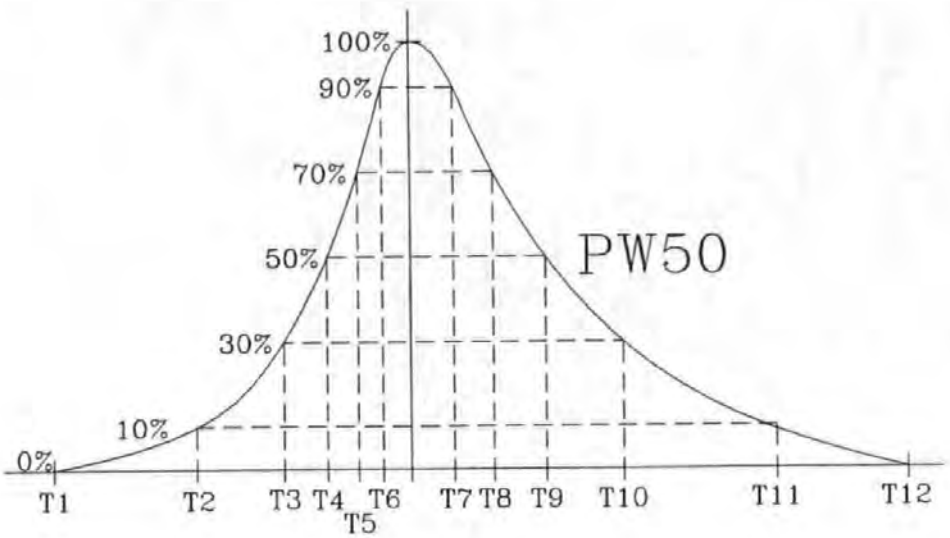


Fig. 3.20. Characterisation of the Isolated Pulses.

F04ATF (NAG, 1987) was used to solve this set (using Crout's factorisation method), thereby determining the coefficients. The values of a and b coefficients for the inductive and MR heads can be found in Appendix D. The isolated pulses produced by the combination of expressions exactly matched that of the reference pulse at the points specified.

3.3.2.3. Signal Amplitude Fluctuations.

The amplitude of the replayed signal displayed on an oscilloscope could be seen to fluctuate or 'bounce'. The four main causes of signal amplitude variations are stated below, and may be derived from equation 3.10 (taken from Appendix B).

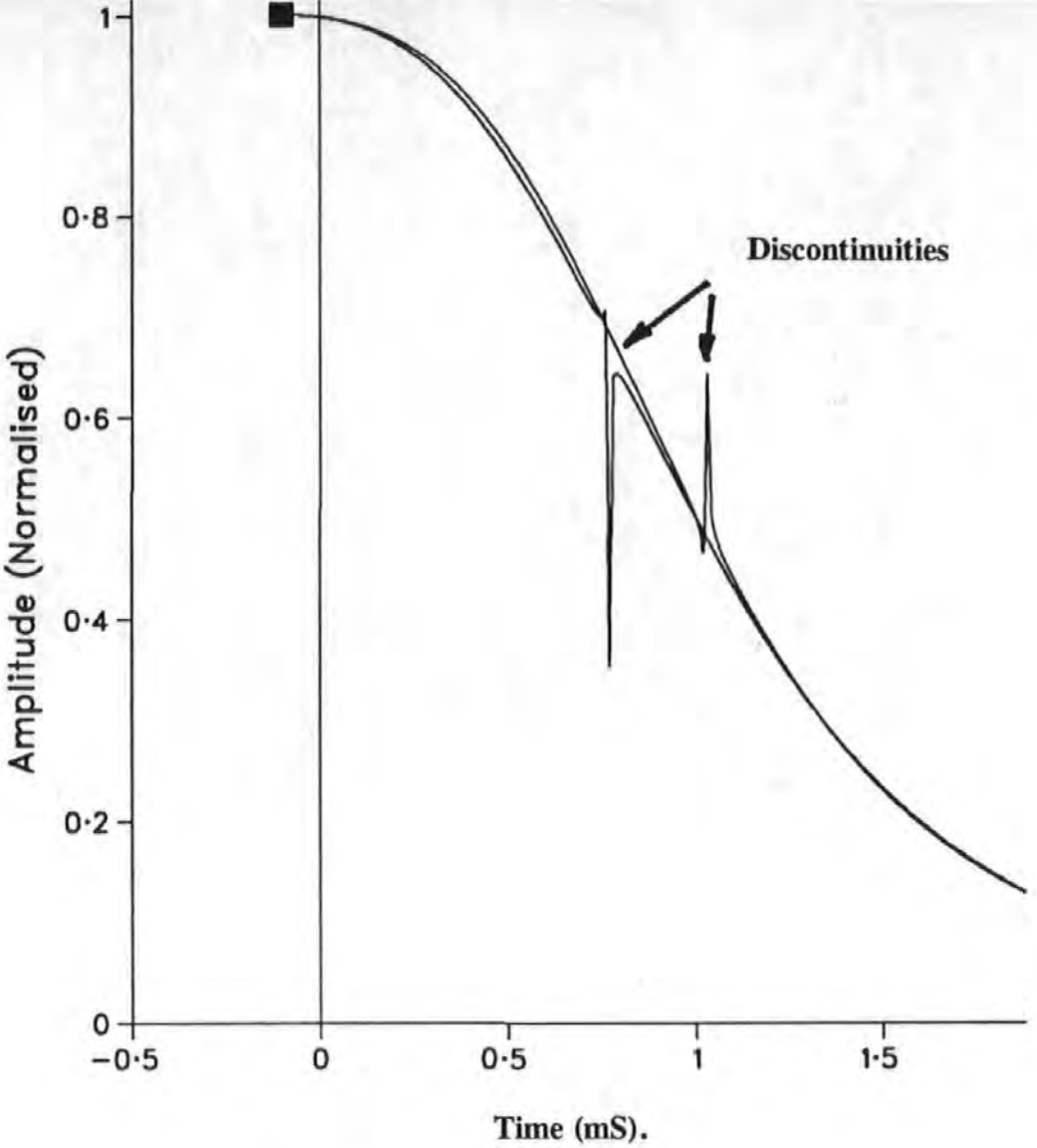


Fig. 3.21. Discontinuities in analytical pulse expressions.

$$e(X) = C_1 Vw.\ k\delta.\ (e^{-kd}).\ \frac{(1-e^{-k\delta})}{k\delta}.\ \frac{\sin(kg/2)}{(kg/2)}.\ \cos(kX) \qquad \text{Equ. 3.10}$$

Spacing Loss — Thickness Loss — Gap Loss

i) Tape speed variations.

The output voltage is proportional to the relative head-to-tape velocity V. Sources of tape speed variations include poorly controlled motor speed, capstan and capstan shaft eccentricity, as well as inconsistent friction in items such as bearings, and between the tape and its pressure pad and tape guides. The time between transitions will also vary as the tape speed varies,

causing problems at the decoding stage. Software based, velocity independent channel decoders that compensate for this effect have been developed (Donnelly, 1989).

ii) Dynamic Lateral Head Displacement.

The output voltage is proportional to the track width w. Lateral Head Displacement (LHD) effectively reduces the track width, attenuating the signal. The linear relationship between track width and signal amplitude results in a 3dB attenuation for a LHD of w/2.

iii) Head-Medium Spacing Loss.

This is one of the most critical parameters in magnetic recording as the amplitude of the reproduced signal is exponentially related to the spacing between the head and the medium, according to,

$$\exp(-kd)$$

where d = head to medium spacing.

k = wavenumber.

Although theoretically $d=0$ for in-contact recording (as should be the case for the compact-cassette), the surface roughness of the medium effectively makes $d>0$. Further increases in the spacing can be caused by inconsistent pressure between the head and tape (exerted by a spring-loaded felt pad in the compact-cassette) and buckled or twisted tape. (Debris between the head and tape will also temporarily increase the spacing loss, but this was classified as a drop-out and treated separately, see section 3.3.2.4.)

iv) Dynamic Azimuth Skew variations.

This is not the static azimuth skew error that can occur when different heads or different machines are used for record and playback. Dynamic azimuth skew variations are caused by the tape 'weaving' across the head in a serpentine manner, due primarily to imperfectly slit tapes, see figure 3.22. Plastic deformation of the tape can also produce an azimuth skew error.

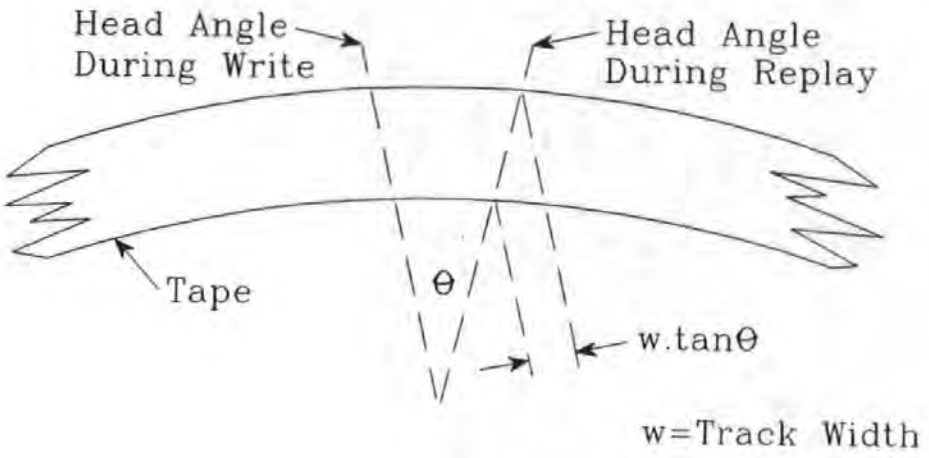


Fig. 3.22. The Effect of Azimuth Variations.

The effect of azimuth skew is linked to the gap loss term in equation 3.10. The gap width obviously does not change. However, if the read head gap orientation is different to the write head gap orientation, the full width of the written transition will not be sensed by the read gap at the same instant. Instead, the flux from the transition will be sensed over a distance $x.\tan\theta$, reducing the sensed rate of change of flux. From Faraday's Law this will lead to a reduction in signal amplitude. The attenuation is frequency dependent, defined by,

$$\frac{\sin(kw\theta/2)}{(kw\theta/2)}$$

where w = read head gap width.

For a 5kHz signal recorded on a compact-cassette (tape velocity of $4.75cm.s^{-1}$ and track width 0.61mm), an azimuth error of just one quarter of one degree gives an attenuation of approximately 3dB.

Rather than attempt to model all these error sources individually, a statistical model of their combined effect was derived. A Bi-Phase-L encoded 'all-ones' signal was recorded onto tape. The peak amplitude of 1028 replayed signal pulses were captured (using a digital waveform analyser) and downloaded to a computer for

statistical analysis, using MINITAB (MINITAB, 1989).

The probability distribution of the amplitude fluctuations was correlated with the Gaussian distribution (Miller, 1988), producing a correlation factor of 0.998. The amplitude fluctuations could therefore be accurately modelled by GWN of the appropriate Standard Deviation and Mean: 0.029 and zero respectively.

3.3.2.4. Drop-Outs.

A drop-out is classified as a short and severe loss of output from the playback head. In order to simulate this error source accurately the level of attenuation and its duration are required. This information was not available. Theoretical models concerning drop-outs do exist, but again require parameters not available. For example, Baker (Baker, 1977) used the physical dimensions of the head and the debris causing the drop-out to calculate the spacing loss and related this to the SNR and bit error rate.

At data rates significantly less than 5kbps the error rate remained essentially constant at $1x10^{-7}$. These errors were attributed to drop-outs, as ISI is negligeable at these data rates, and as the comparator threshold levels could be raised considerably higher than the noise level with no effect on this base error rate. Drop-outs were therefore modelled using a Uniform probability distribution (of $1x10^{-7}$), attenuating the isolated pulse by 20dB.

3.3.2.5. Lateral Head Displacement.

In the compact-cassette system the head mount was modified, allowing the head to be physically displaced. The model had similar capabilities, allowing the introduction of static Lateral Head Displacement (*LHD*), in controlled and measured amounts, in addition to the dynamic LHD discussed in section 3.3.2.3. This section describes how this was implemented.

Track misregistration attenuates the replay signal in

proportion to the reduction of effective flux linking the magnetic circuit of the head (Abbot et al., 1988). The magnitude of this attenuation is therefore not only dependent on the distance the head is displaced, but also on the dimensions and geometry of the head and track format.

Additionally, in a parallel track system, large values of LHD may result in the head of one track linking with flux from an adjacent track. For a standard compact-cassette system the amount of displacement normally encountered is insignificant, when compared to the dimensions of the tracks. As track dimensions decrease (to increase areal bit densities) LHD becomes more of a problem. In order to investigate significant amounts of LHD, both the compact-cassette system and its model were modified to be able to introduce sufficient LHD for the n^{th} head to align with data recorded by the $(n-1)^{th}$ head.

Figure 3.23 shows a single written track with its associated read head, which as indicated may be a different width to the write head. The written track is shown divided into 3 regions. The central region, w wide, corresponds to the write head gap core width. The total written track width is known to be wider than w due to side-writing effects (Lindholm, 1977, and Ichiyama, 1977).

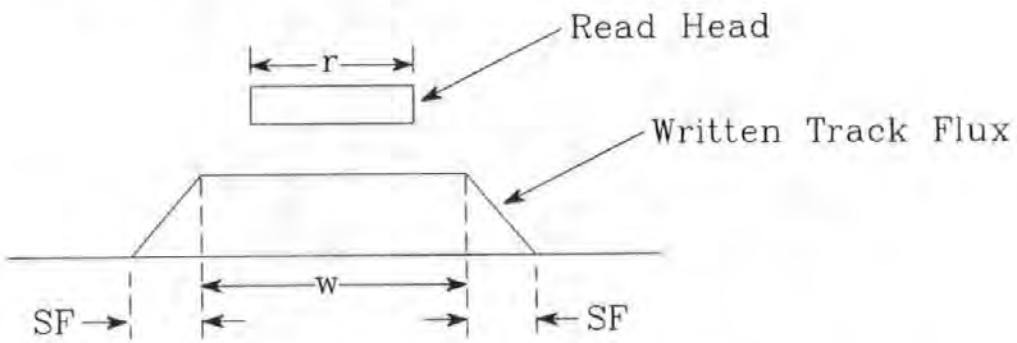


Fig. 3.23. Relationship between Written Track and Read Head.

Side-reading effects also exist (van Herk, 1977) and are exponentialy related to the distance between the read head and written track. Also, from van Herk, the recorded transition width increases

proportionally with the distance, effectively 'smearing' the off-track sensed flux in the time-domain. The total effect of these three side-field related phenomena was simplified in the model to a linear reduction in signal amplitude with distance, from the normalised maximum to zero in a distance of SF, as shown.

Figure 3.24 shows two written tracks (n and n-1) with their associated read heads. The read heads are shown with increasing amounts of displacement, m, from correct alignment, figure 3.24(a), to displacement equal to the track separation, figure 3.24(g). Figure 3.24(d) illustrates the worst-case situation for the n^{th} head. On figure 3.24(d), six hatched areas are indicated, corresponding to the amount of flux linking a head. For example, the area indicated by $b_{n:n-1}$ corresponds to the amount of the $(n-1)^{th}$ track's side field that the n^{th} head links with. Figure 3.24(g) shows the n^{th} head perfectly aligned with the $(n-1)^{th}$ track, and the $(n-1)^{th}$ head linking with no flux (effectively off the tape).

If f(n) and f(n-1) represent the optimal reproduction of tracks n and n-1 respectively, and f'(n) and f'(n-1) the actual signals detected by replay heads n and n-1 respectively, then:

$$f'(n) = \frac{(a_{n:n}+b_{n:n})f(n)}{w} + \frac{(a_{n:n-1}+b_{n:n-1})f(n-1)}{w}$$

Eqn. 3.11

Like all magnetic fields, side fringing fields are affected by spacing loss, i.e. attenuation by exp(-kd). The effects of side fringing fields are therefore critically dependent on d and k. For wide write widths and low frequencies (as used in the compact-cassette system), their effects are negligible and may be ignored. Their inclusion in the model allows narrow track systems, where their effects may be considerable, to be investigated. Therefore, during simulation of the compact-cassette system the side-fringing field width was set to zero. As the read head width equals the write head width, the derivation of coefficients a and b simplifies to,

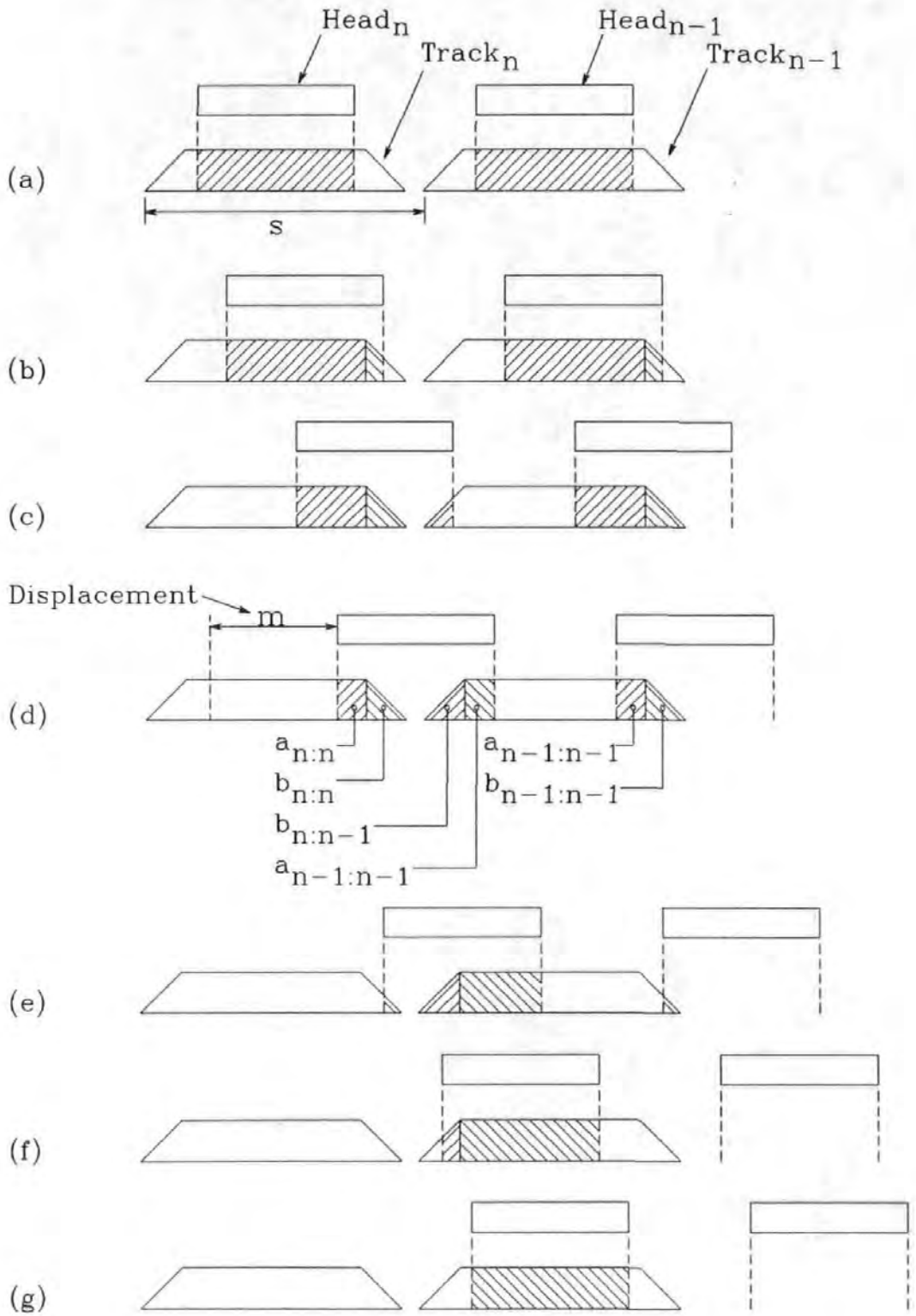


Fig. 3.24 The Effect of Track Misregistration on the amount of Flux linking with the Read Head.

$b = 0$

$a_{n:n} = (w-m)$, $a_{n:n-1} = 0$ $\quad 0 < m < (s-w)$

$a_{n:n} = (w-m)$, $a_{n:n-1} = (m+w-s)$ $\quad (s-w) < m < w$

$a_{n:n} = 0$, $a_{n:n-1} = (m+w-s)$ $\quad w < m < s$

In the model, LHD was simulated by calculating the values of the a and b coefficients (using the track format dimensions and the displacement) and using equation 3.11 to introduce the required level of cross-talk.

3.3.2.6. Data Skew between Tracks.

Each track was processed as a separate bit-serial data channel, therefore the relationship between the tracks, in terms of alignment of data, had no effect on the performance of the system (under normal operating conditions). However, when the head displacement is large enough for adjacent tracks to interfere, the alignment of data between tracks becomes very important. To accommodate this in the model, the amount of data skew for each track was specified.

As the LPS waveforms were assembled using an array, the desired amount of data skew was introduced by adjusting the array pointer that specifies the location from which data was output.

3.3.2.7. Addition of Medium Noise.

The Electronic Noise in the compact-cassette system was so large as to prevent measurement of the Medium Noise. As most state-of-the-art systems are medium noise limited, this error source was included in the model for completeness, even though it was set to zero during simulation of the compact-cassette system.

As it was not possible to measure the magnitude of the Medium

Noise, the level of medium noise introduced was that which would produce the same amount of Electronic Noise, measured at the GXO detector comparators. The addition of GWN with a Standard Deviation of 1.82×10^{-6} produced comparable values, as shown below.

	SD of Electronic Noise, measured at comparators.	SD of noise measured at the comparators when Media Noise with SD=1.82×10^{-6} added.
Peak Circuit	1.86×10^{-3}	1.93×10^{-3}
Polarity Circuit	4.1×10^{-4}	3.97×10^{-4}

Using the RMS value of the noise and the simulated 5kbps Bi-Phase-L encoded PRBS waveform, the SNR of the simulated system was calculated to be:

$$20\log \frac{8.01 \times 10^{-4}}{1.82 \times 10^{-6}} = 52.9 \text{ dB}$$

This is a realistic value for a compact-cassette system (AGFA, 1973). The medium noise was added to the waveform prior to the Head Amplifier.

Medium or Particulate noise power is dependent on the number of magnetic particles sensed by the replay head. This is usually assumed to fit a Poisson distribution (Mallinson, 1987b, 5.2.3). Using GWN (which has a Normally distribution) is therefore an approximation, but one that is often made (e.g. Abbot et al., 1988). Thin-Film media noise is strongly dependent on the signal, and would require a completely different model (Wood, 1987).

3.3.3. Model of the Replay Electronics.

The replay electronics was composed of the Head Amplifier, Gated Cross-Over detector and Link Adapter Interface Board. The Head

Amplifier and GXO's analogue electronics were assembled from circuit elements whose primary tasks were to amplify and modify the frequency content of the signal. These elements were therefore modelled as digital filters. The operation of the digital electronics of the GXO detectors and the Link Adapter Interface Board was described directly in *occam*.

3.3.3.1. Z Domain Description of Analogue Circuit Elements.

The frequency modifying effects of the head amplifier and analogue elements of the GXO were modelled using digital filters based on 1st Order Bandpass Butterworth analogue filters. This approximation was made because to model the various elements accurately would require filters of higher order. These take more time to compute, due primarily to the increase in the number of floating point multiplications required (a time consuming operation).

The frequency response of the filters were determined by their coefficients and the effective sampling frequency. These were calculated at run-time from the two -3dB corner frequencies. An indirect design methodology (e.g. Terrel, 1980), using a prototype continuous filter together with the Bilinear Transform, was chosen. There are three stages in this design process:

i) Determine Filter Specification.

This is dependent on the circuit elements being modelled, and is specified by the upper and lower corner frequencies (f_u and f_l). These frequencies need to be converted into their angular frequency forms, and then Pre-Warped to compensate for the frequency warping effect of the Bilinear Z-Transform, thus:

$$w_c = \frac{2}{T}\tan\left(\frac{T}{2}\cdot\frac{f_c}{2\pi}\right) \qquad \text{Equ. 3.11}$$

ii) Design Continuous Filter.

The Normalised low-pass 1st order Butterworth filter is defined by:

$$G(S) = \frac{1}{(S+1)} \qquad \text{Equ. 3.12}$$

It is transformed into its bandpass form using the substitution:

$$S \dashrightarrow \frac{(S^2+(w_{cl}/w_{cu}))}{S(w_{cu}-w_{cl})} \qquad \text{Equ. 3.13}$$

iii) Transform from S-plane to Z-plane representation.

This was done using the Bilinear Z-Transform, defined by:

$$S = \frac{2}{T}.\frac{(Z-1)}{(Z+1)} \qquad \text{Equ. 3.14}$$

This results in an equation in the form:

$$G(Z) = \frac{a(1-Z^{-2})}{1+bZ^{-1}+cZ^{-2}} = \frac{Y(Z)}{X(Z)} \qquad \text{Equ. 3.15}$$

As the Z^{-1} terms represents a delay of one sample period, the above equation can be readily rewritten in the sampled time domain, thus:

$$y_n = a(x_n - x_{n-2}) - by_{n-1} - cy_{n-2} \qquad \text{Equ. 3.16}$$

This is the equation used in the ***occam*** programme to implement the filter. It requires 3 floating point multiplications and subtractions per sample. This takes approximately 1300 CPU cycles or 86μS for a 15MHz T414 ***transputer***.

3.3.3.2. Head Amplifier.

The compact-cassette Head Amplifier was a two stage design: two low-pass gain blocks, AC coupled. This was approximated to a bandpass filter in the model, with f_l and f_u being determined by the coupling stage and the first stage of amplification respectively, see

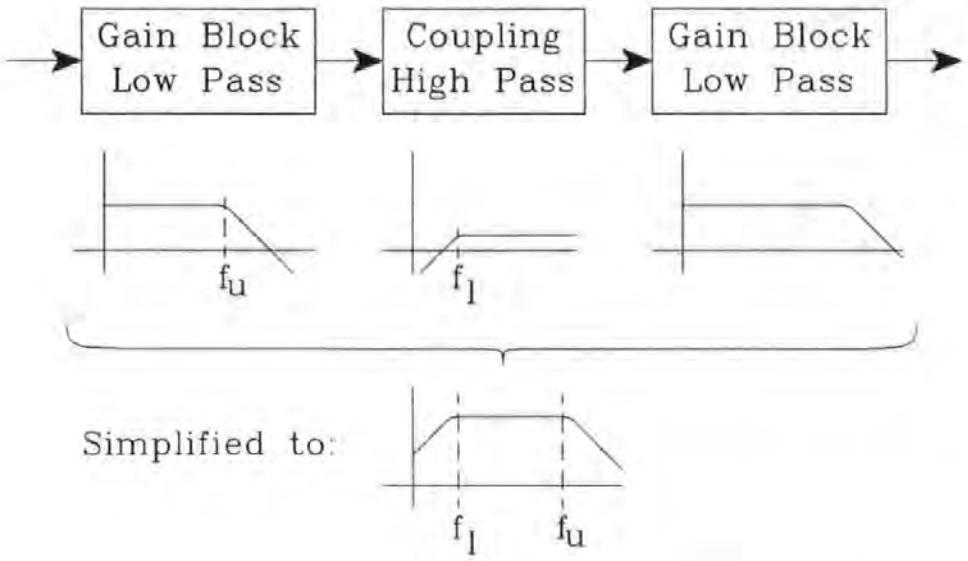


Fig. 3.25. Approximation of Head Amplifier by Bandpass Filter.

figure 3.25.

3.3.3.3. Gated Cross-Over Detector.

The model of the GXO detector was divided into the same three distinct functions as the hardware was (detailed in section 2.3.3), ie:

i) Gating Signal or Peak Centre Detection.

This was performed in the compact-cassette system by an operational amplifier based differentiator, high-pass coupling stage, and a comparator with variable hysteresis. The -3dB frequency of the coupling stage was 100Hz. As the preceding stage (the differentiator) had already attenuated the signal at 100Hz by 17dB, the effect of the coupling stage was not included in the model. The differentiator was modelled as a bandpass filter (as detailed in section 3.3.3.1), with $f_l = 10$kHz and $f_u = 50$kHz.

The variable hysteresis comparator was modelled easily in ***occam*** by:

```
IF
  (output=HIGH) AND (input < negative.threshold)
    output:=LOW
  (output=LOW) AND (input > positive.threshold)
    output:=HIGH
  TRUE -- else,
    SKIP -- output remains unchanged
```

The threshold value was read-in at run-time from the parameter 'fold' (see section 3.3). A value of 40mV was used during compact-cassette system simulation. Transitions of the digital output signal corresponded to centres of peaks in the analogue waveform. This formed the Gating signal.

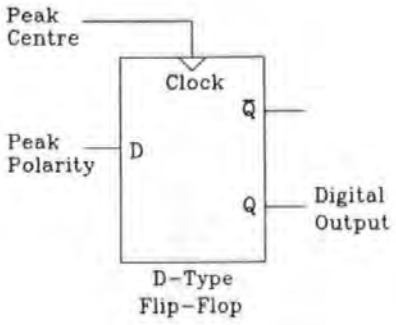
ii) Peak Polarity Discrimination.

This was performed in the compact-cassette system by an operational amplifier based low-pass gain block, high-pass coupling stage, and comparator with variable hysteresis. The low-pass gain block and the high-pass coupling stage were modelled as a bandpass filter (as detailed in section 3.3.3.1), with f_l=102Hz and f_u=6.8kHz. The comparator is as for the Peak Detector, with the threshold level set at 1.5mV during compact-cassette system simulation. The level of the output signal corresponded to the polarity of the peaks in the analogue waveform.

iii) Digital Output.

It is necessary to combine the timing information of the Peak Detector signal with the data of the Polarity signal. Figure 3.26 shows how this was achieved in the compact-cassette system, and how it was modelled.

Once the basic elements of the GXO detector model were written, they were 'connected' together in much the same way as the hardware, see figure 3.27. Instead of electrical signals passing along wires, numeric values representing the magnitude of the signals



(a)

```
IF
  peak.signal <> old.peak.signal
    output := polarity.signal
  TRUE -- ELSE
    SKIP -- Output unchanged
```

(b)

Fig. 3.26. Combining the Timing and Polarity Information,
(a) Hardware, (b) Software.

were passed down ***occam*** channels.

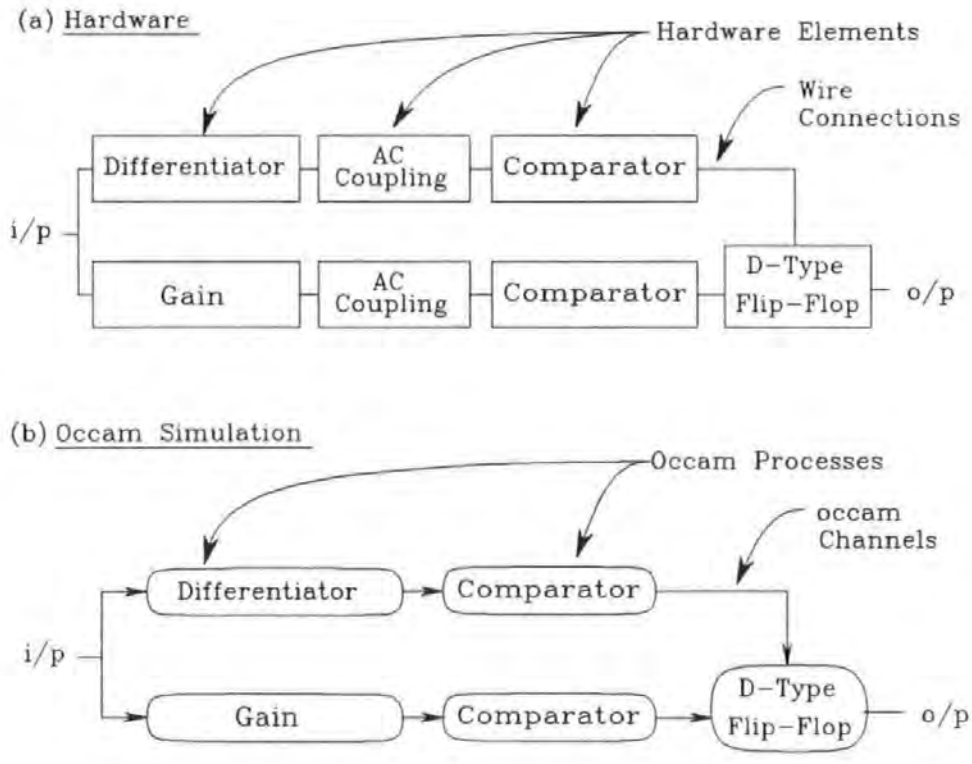


Fig. 3.27. GXO Detector (a) Hardware Block Diagram
(b) ***occam*** Process Diagram.

3.3.3.4. Addition of Electronic Noise.

All Electronic circuits generate noise. There are two main electrical noise sources (e.g. Connor, 1982): Thermal Noise; and Shot Noise. Both are due to the random motion of the electrons as they move through the conductor. Thermal Noise relates this randomness to the temperature of the conductor, whilst Shot Noise relates the random arrival of electrons to the magnitude of the average current. Both these noise sources have approximately uniform spectral densities for the frequencies of interest, and were therefore modelled by GWN (see section 3.3.1).

Although the majority of the electronic noise is usually generated by the first stage of amplification (Mallinson, 1987, section 5.2.1) (as it is amplified by the total gain of the system), its effect is most evident at the level thresholding comparators of the GXOs. When the signal is near to the switching threshold, a small amount of noise may be sufficient to cause a false trigger, as illustrated in figure 3.28.

Electronic noise was added at the comparator stage of the Peak Detector and Polarity Discriminator. To characterise the noise, a waveform analyser was used to digitise the noise between the two input pins of each comparator. The digitised waveforms were downloaded to a computer for statistical analysis (using MINITAB). The probability distribution of the noise was correlated with the Normal distribution (Miller, 1988). The correlation factor was 1.000 (i.e. to three decimal places) for both waveforms. The electronic noise could therefore be accurately modelled by GWN. The Standard Deviations were 1.86×10^{-3} and 4.1×10^{-4} for the Peak Detector and Polarity Discriminator Comparators respectively.

3.3.3.5. Link Adapter Interface Board.

The data from the model must be in exactly the same format as the data from the compact-cassette system, as the same software is used to process both data streams (see figure 3.1). The sampled data

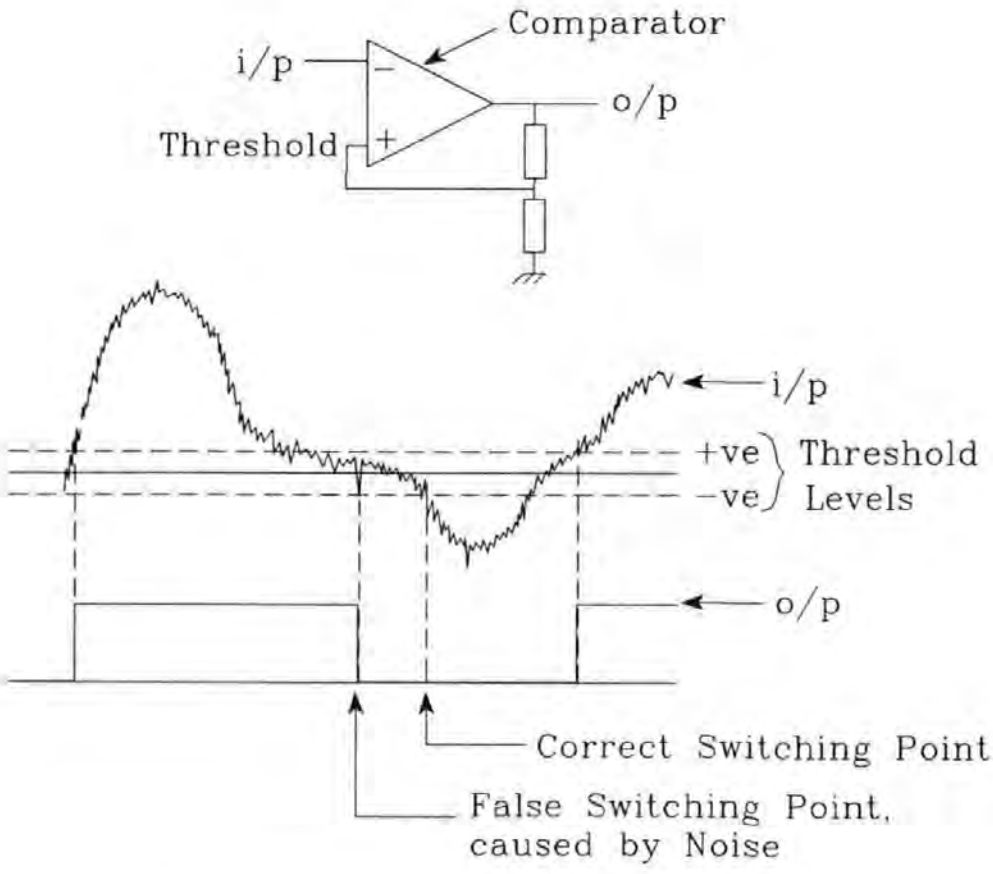


Fig. 3.28. The Effect of Noise on the Operation of a Comparator.

stream of the model must therefore be transformed to an Event data stream. The input data stream from all channels was monitored for changes in data (i.e. events). When an event was detected, the data from all the channels were combined to form a new data word, and the time of its occurrence was reconstructed. The new data word was stored in one array and the time of its occurrence in a second. These two arrays constituted the interface between the model and the real-time software.

3.4. Lateral Head Displacement Compensation Scheme.

In section 3.3.2.5, equation 3.11 was developed defining the received signals, f'(n), in terms of the optimally reproduced signals, f(n).

Barbosa (Barbosa, 1990) developed expressions for the signals detected by M heads reading N tracks (N not necessarily equal to M). Barbosa expressed the relationship in terms of mappings between a 'data space' and an 'observation space' (Barbosa, 1989) (compare with 'optimally reproduced' and 'received' signals), to produce a linear processor that removes the cross-talk from interfering signals. A similar line of reasoning is applied to the compact-cassette system.

Figure 3.29 restates the nomenclature developed in section 3.3.2.5, whilst equation 3.17 is equation 3.11 rearranged and with zero width side-fields.

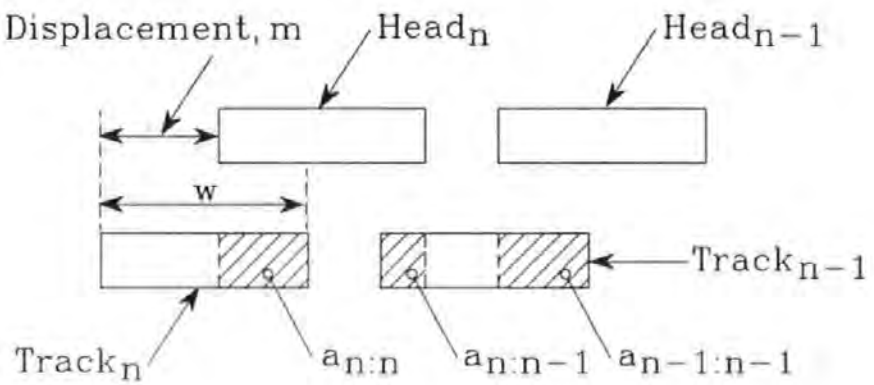


Fig. 3.29. Nomenclature used to describe the Effects of LHD for Compact-Cassette System.

$$f(n) = \frac{w}{a_{n:n}}.f'(n) - \frac{a_{n:n-1}}{a_{n:n}}.f(n-1) \qquad \text{Equ. 3.17}$$

This states that the optimally reproduced signal from n^{th} track is equal to a scaled amount of the received signal from the n^{th} track minus a proportion of the optimaly reproduced signal from the $(n-1)^{th}$ track. This is of little use as no signals are optimally reproduced when the head is displaced. In a practical tape system, a simplification may be made. In an N track tape system experiencing LHD either the 1^{st} head or the N^{th} head (depending on the direction of displacement) will not experience interference from adjacent tracks. If the direction of displacement results in the n^{th} head moving towards the $(n-1)^{th}$ track, then for the 1^{st}

head, equation 3.17 reduces to,

$$f(1) = \frac{w}{a_{1:1}} \cdot f'(1) \qquad \text{Equ. 3.18}$$

As w is known and $a_{1:1}$ can be calculated, the optimally received signal f(1) can be reconstituted from the received signal f'(1). This result can now be used in equation 3.17 to calculate f(2). This process may be repeated across the width of the tape, effectively removing the effect of the LHD from all head signals.

This scheme requires a knowledge of the track format dimensions and the LHD present (both magnitude and direction). The displacement should ideally be derived from the read signals, as this would remove the problem of the tracks having been written under the influence of LHD. If data were recorded at different data rates on different tracks, the ratio of replayed fundamental frequency magnitudes would indicate the LHD. Although a similar method of displacement measurement has been successfully used in the laboratory environment (Su, 1990), it would considerably increase the complexity of a commercial system. Note, these ideas would be very simple to investigate using the model.

From equation 3.10 the magnitude of the replayed signal is linearly related to the width of track read, and is therefore dependent on the LHD. This is only valid for LHD $<$ (w - s) (i.e. displacements within the inter-track guard-band), as for greater displacements an adjacent track will interfere. However, as highlighted above, one of the peripheral heads will not be corrupted by an adjacent track. The problem is therefore reduced to calculating which peripheral track is not being corrupted by an adjacent track, and using the reduction in signal amplitude from this head to calculate the LHD. This technique benefits from narrow inter-track guard-bands as the corruption of signals from adjacent tracks would allow the direction to be calculated for small values of LHD. Reducing the guard-band widths increases areal packing density and is therefore desirable.

From visual comparisons of waveforms from heads simply

attenuated by LHD, and waveforms from heads corrupted by an adjacent track, the signal content appears significantly different. The suggestion is therefore made that a suitable signal processing technique could be developed to differentiate between the two waveforms.

If it is not possible to extract the necessary information regarding LHD from the standard head signals, the modified read head shown in figure 3.30 may be used to provide the information.

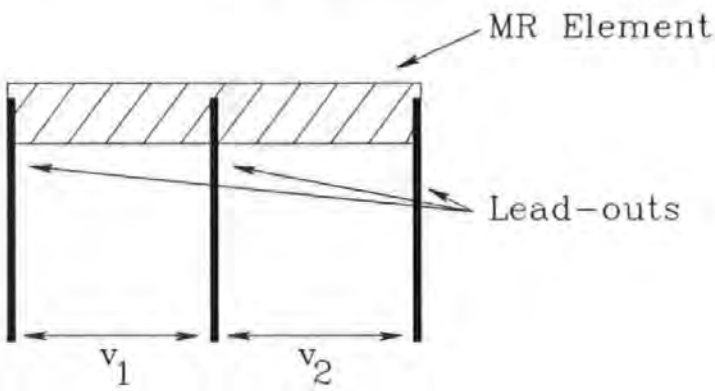


Fig. 3.30. Magneto-Resistive Differential Read Element.

Figure 3.30 shows an MR Differential Element, although an equivalent inductive version is also possible. The signal v_1+v_2 is the same as for a full width MR stripe. However, the relative magnitude of the signals v_1 and v_2 could be used to determine the direction and magnitude of the LHD.

The compensation scheme described above uses the signal from the n^{th} head reading the $(n-1)^{th}$ track to estimate the signal from the $(n-1)^{th}$ head. Depending on the directions of the coil windings, there may be an inversion that is not apparent from the undisplaced read signals alone, see figure 3.31 (where a dot notation is used to indicate coil directions).

As can be seen in figure 3.31 although a reversed coil direction results in a reversed magnetisation pattern, if the coil winding directions are consistent between record and replay, the replayed waveforms are phase correct. If the replay coil is reversed with respect to the record coil, then the signal will be inverted.

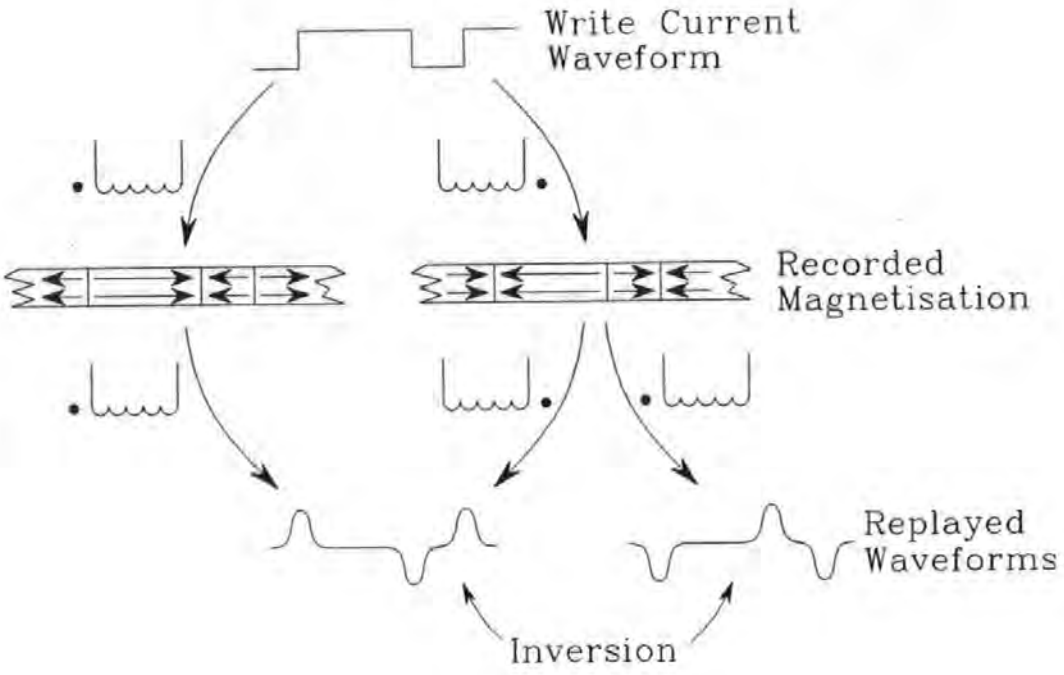


Fig. 3.31. The relationship between read signal and recorded magnetisation.

This must be taken into account in the compensation scheme or the scheme will not simply fail to work, it will degrade performance.

To investigate these ideas in a practical system, components of the compact-cassette system and model were combined to produce the system illustrated in figure 3.1(c). A commercial digital waveform analyser was used to simultaneously digitise the signals from two of the compact-cassette's channels. These digitised signals were then transferred via an RS232 serial link to the IBM PC for storage on its hard disc.

The data captured by the waveform analyser represents a sampled data stream, and was therefore incorporated into the model very easily. The only extra ***occam*** code that needed to be written was that which ensured the format of the data complied with ***occam***'s (strict) floating point format. The compensation scheme was applied to these waveforms before conversion to TTL type waveforms by the GXO detectors. The signal processing software was the same as that used in the compact-cassette system and model. The

only modification necessary was a reduction in the sampling rate to 50kHz to match that of the waveform analyser (sampling at its highest rate). This in turn necessitated a similar reduction in cut-off frequencies for the digital filters to ensure an adequate sampling rate was maintained (to avoid the effects of aliasing).

3.5. Summary.

This chapter has described the software used in 3 magnetic recording channels, written in ***occam***. The first channel used software to generate data and encode it suitably for recording onto a compact-cassette. During replay software was used to decode the data, and to check for and classify errors before filing on the IBM PC's disc. The second channel was formed by replacing the hardware components of the first system with software models. This produced a complete model of a multiple-track digital magnetic recorder. The third channel re-used components of the previous two and was used to investigate the performance of the proposed Lateral Head Displacement compensation scheme.

A PRBS generator was used to generate the test data. It was programmable in terms of sequence length (between 7 and 32767 bits) and in terms of relationship between sequences (data staggered or un-staggered), and could generate up to 32 such sequences simultaneously for use in a multiple-track system. The characteristics of this data stream were matched to that of the recording channel by a Bi-Phase-L channel encoder.

In the compact-cassette system this data was output at timed intervals via one of the ***transputer***s Links to the write amplifier for recording onto tape. When replayed, the data were read back into the ***transputer***, and stored in an array together with the time they were read (to an accuracy of approximately 3.5μS). The data from these areas were separated and distributed for processing on a 'track by track' basis. A Bi-Phase-L channel decoder was followed by an error checking and classification process. The error classification scheme devised provided more detailed information

regarding errors than the raw bit error count used by many Researchers.

In the model, the Bi-Phase-L encoded PRBS data were channelled directly to the Linear Superposition process, as the write process was assumed to be perfect. Analytical expressions were derived that accurately described the shape of the isolated pulses produced by the inductive and magneto-resistive heads. Based on measurements taken from the compact-cassette system, sources of electronic and medium noise, as well as waveform amplitude fluctuations (including drop-outs) were incorporated into the model. A general purpose GWN generator was developed for use by these error sources. By calculating the flux linkage between the displaced heads and the recorded tracks, the model also allowed Lateral Head Displacement to be simulated. Digital filters were used to model the frequency modifying components of the head amplifier and GXO detectors, whilst the digital components were described directly in ***occam***.

Components of the model were used to investigate a LHD compensation scheme. The scheme devised was the inverse of that used to model LHD. From a knowledge of the dimensions of the track format and LHD, the amount of cross-talk from an adjacent track was calculated. An estimate of the uncorrupted signal was produced by subtracting the correct amount of the relevant adjacent track signal.

The software developed formed a complete model of the recording process - from data generation to error detection and classification. As a significant proportion of the code was common to all three system, a high level of consistency between them was assured. The modular style of programming used to design the ***occam*** processes resulted in a very flexible model.

3.6. References for Chapter 3.

ABBOTT, W.L., Cioffi, J.M., Thapar, H.K., Offtrack Interference and Equalisation in Magnetic Recording. IEEE Trans. on Magnetics, Vol. MAG-24, No. 6, November 1988.

AGFA-GEVAERT, Recording with Compact-Cassettes, 1st Edition, Published by AGFA-GEVAERT, 1973.

ATKIN, P., Performance Maximisation. INMOS Technical Note 17, 72-TCH-017-00, March 1987.

BAKER, W.R., A Dropout Model for a Digital Tape Recorder. IEEE Trans. on Magnetics, Vol. MAG-13, No. 5, September 1977.

BARBOSA, L.C., Maximum Likelihood Sequence Estimators: A Geometric View. I.E.E.E. Trans. on Information Theory, Vol 35, No. 2, March 1989.

BARBOSA, L.C., Simultaneous Detection of Readback Signals from Interfering Magnetic Recording Tracks using Array Heads. IEEE Trans. on Magnetics, Vol. MAG-26, No. 5, September 1990.

BURNS, A., & Wellings, A.J., Occam's Priority Model and Deadline Scheduling. 7th Occam Users Group & Int. Workshop on Parallel Programming of Transputer based Machines, Grenoble, September 1987.

BURNS, A., Programming in occam 2. Addison-Wesley Pub. Ltd., 1988

CARLINI, D.U., & Villano, U. A Simple Algorithm for Clock Synchronisation in Transputer Networks. Software: Practice and Experience, John Wiley & Sons Ltd., Vol. 18(4), April 1988.

CONNOR, F.R., Noise. Edward Arnold (Pub.) Ltd., London, 1982.

DOWSING, R.D., Simulating Hardware Structures in occam. Software & Microsystems, Vol. 4, No. 4, August 1985.

DJAHANGUIR, A.H. & Geffroy, J.C. Use of occam for Validation of Distributed Discrete Event Driven Simulation. Proceeding of the 10th occam User Group Technical Meeting, Enschede, Netherlands. Published by IOS, Amsterdam. 1989

GOOD, B., Private Communication. Computing Dept., Polytechnic South West, 1989.

GORDON, G., System Simulation. Prentice-Hall Inc., New Jersey, 1978.

ICHIYAMA, Y., Analytic Expressions for the Side Fringe Field of Narrow Track Heads. IEEE Trans. on Magnetics, Vol. MAG-13, No. 5, January 1977.

INMOS Ltd., Transputer Reference Manual. Prentice Hall International (UK) Ltd., 1988 (a).

INMOS Ltd., Occam 2 Reference Manual. Prentice Hall International (UK) Ltd., 1988 (b).

JACKSON, T.J., Mapps, D.J., Ifeachor, E.C. & Donnelly, T. A Real-Time Transputer-Based System for a Digital Recording Data Channel.", Microprocessing and Microprogramming Vol. 25, pp 281-286, 1989.

KNUTH, D. E., The Art of Computer Programming, Volume 3: Seminumerical Algorithms. Addison-Wesley Pub. Co., 2nd Edition, 1981.

LINDHOLM, D.A., Magnetic Fields of Finite Track Width Heads. IEEE Trans. on Magnetics, Vol. MAG-13, No. 5, September 1977.

LOZE, M.K., Middleton, B.K., Ryley, A, and Wright, C.D., A Comparison of Various Methods for Characterising the Head-Medium Interface in Digital Magnetic Recording. IEEE Trans. on Magnetics, Vol. MAG-26, No. 1, January 1990.

MACWILLIAMS, F.J., and Sloane, J.A., Pseudo-Random Sequences and Arrays. Proc. IEEE., Vol. 64, No. 12, December 1976.

MACKINTOSH. N.D. The choice of a Recording Code. I.E.R.E. Conf. Proc. No. 43, Southhampton 1979 (a).

MACKINTOSH, N.D., A Superposition-Based Analysis of Pulse-Slimming Techniques for Digital Recording. IERE Conf. Proc. No. 43. Southampton 1979, (b).

MALLINSON, J.C., & Steele, C.W. Theory of Linear Superposition in Tape Recording. IEEE Trans. on Magnetics, Vol. MAG-5, December 1969.

MALLINSON, J.C., The Foundations of Magnetic Recording. Academic Press, Inc., USA, 1987 (a).

MALLINSON, J.C., Chapter 5, Magnetic Recording, Vol. I: Technology, Series Editors C. D. Mee, and E. D. Daniels, McGraw-Hill, Inc. USA, 1987 (b).

MIDDLETON, B.K. and Wisely, P.L., Pulse Superposition and High-Density Recording. IEEE Trans. on Magnetics, Vol. MAG-14, No. 5, September 1978.

MILLER, R.B., Minitab Handbook for Business and Economics. PWS-Kent Publishing Co., Boston, MA, 1988.

MINITAB, MINITAB Reference Manual. MINITAB Inc., USA, 1989

NAG, The NAG Fortran Library Manual. The Numerical Algorithms Group, Oxford, UK, 1987.

NEVISON, C., Discrete Event Simulation using occam. Proceeding of the 10th occam User Group Technical Meeting, Enschede, Netherlands, 1989. Published by IOS, Amsterdam.

POUNTAIN, D., A Tutorial Introduction to occam Programming. BSP Professional Books, London, 1987.

SU, J.L., Ju, K., Lo, J., & Countryman, G. Side Fringing of Thin Film Heads with Pole Trimming. IEEE Trans. on Magnetics, Vol. MAG-16, No. 5, September 1990.

SZCZECH, T.J., The use of Equations for the Field Components of a Thin-Film Head for Calculating Isolated Pulse Output. IEEE Trans. on Magnetics, Vol. MAG-16, No. 5, September 1980.

TELLAGRAF, CA-TellaGraf User's Guide. Computer Associates Int., Inc., USA, 1987.

TERREL, T.J., Introduction to Digital Filters. MacMillan Pub. Ltd., UK, 1980.

VAN HERK, A., Side Fringing Fields and Write and Read Crosstalk of Narrow Magnetic Recording Heads. IEEE Trans. on Magnetics, Vol. MAG-13, No. 4, July 1977.

WELCH, P.H., Managing Hard Real-Time Demands on Transputers. 7th Occam Users Group & Int. Workshop on Parallel Programming of Transputer based Machines, Grenoble, September 1987.

WOOD, R., Jitter VS Additive Noise in Magnetic Recording: Effects on Detection. IEEE Trans. on Magnetics, Vol. MAG-23, No. 5, September 1987.

4. Results and Discussions.

4.1. Experimental Procedures and Operating Conditions.

The results from the compact-cassette system presented in this chapter were gathered using the following operational procedures.

- The surface of the recording head, pinch roller and capstan were cleaned with Iso-Propyl Alcohol and a cotton bud at regular intervals between replays and always before recording.
- The head azimuth angle and lateral displacement were aligned using a commercial Test Tone Compact-Cassette.
- Recordings were not made on the first 50cm of tape, as this lead-in portion is recognised as being prone to mechanical damage and debris.
- TDK AD90 Compact-Cassettes were used. These are inexpensive IEC/Type I cassettes that use gamma-Ferric Oxide tape.

Unless otherwise stated, the following system parameters were used in the acquisition and processing of the data presented in this chapter.

- Data was staggered by one code bit with respect to adjacent tracks.
- Data was generated using Pseudo-Random Binary Sequences of length 7, at 5k bits per second per track.
- Bi-Phase-L channel code was used with a recording current of $\pm 330\mu A$.
- The error classification scheme used a minimum 'good' sequence length of 5, and a maximum 'bad' sequence length of 7 (see section 3.2.2.4).

Several graphs presented in this chapter plot the error rate on a logarithmically scaled y axis. A value of zero cannot therefore be represented. On such graphs a "Zero Error Rate" level is indicated, corresponding to a rate of 1/total bits. Points plotted at

this rate correspond to an error rate of less than 1 in the total number of bits, but by an amount unknown (further testing would be required for a more accurate figure).

4.2. Accuracy of the Isolated Pulse Models.

The shape of the analytically generated isolated pulses used to model the output from the inductive and MR heads are shown in figures 4.1 and 4.2. The 13 points used to specify the shape of the pulses (see section 3.3.2.2) are indicated by crosses. Also shown are typical pulses captured using an oscilloscope and plotted onto paper.

The differences in shape between the analytical and measured pulses illustrates the variability between individual pulses, rather than the accuracy of the technique used to derive the equations that determine their shape. The analytical pulses can be seen to fit at the points specified: a result of the technique used to derive the analytical equations. A more accurate fit would require the specification of more points. However, the numeric technique used to derive the equation would require the use of a higher order equation to fit more points. This would increase the probability and severity of discontinuities (see section 3.3.2.2). The use of thirteen data points proved to be a satisfactory compromise.

The pulses from the MR head varied less than those from the inductive head. This was attributed to the MR head reading a track considerably wider (12 times) than the sensing element (having been written by the inductive head), and therefore not affected by dynamic LHD.

If the success of the pulse fitting technique was based solely on how accurately the analytical pulses fitted the measured pulses at the specified points, the technique would be deemed 100% successful. In practice, there are two fundamental limitations:

i) The technique requires a large amount of human intervention to produce reliable results. All of the equations produced by the technique used, contained at least one severe discontinuity. Only by

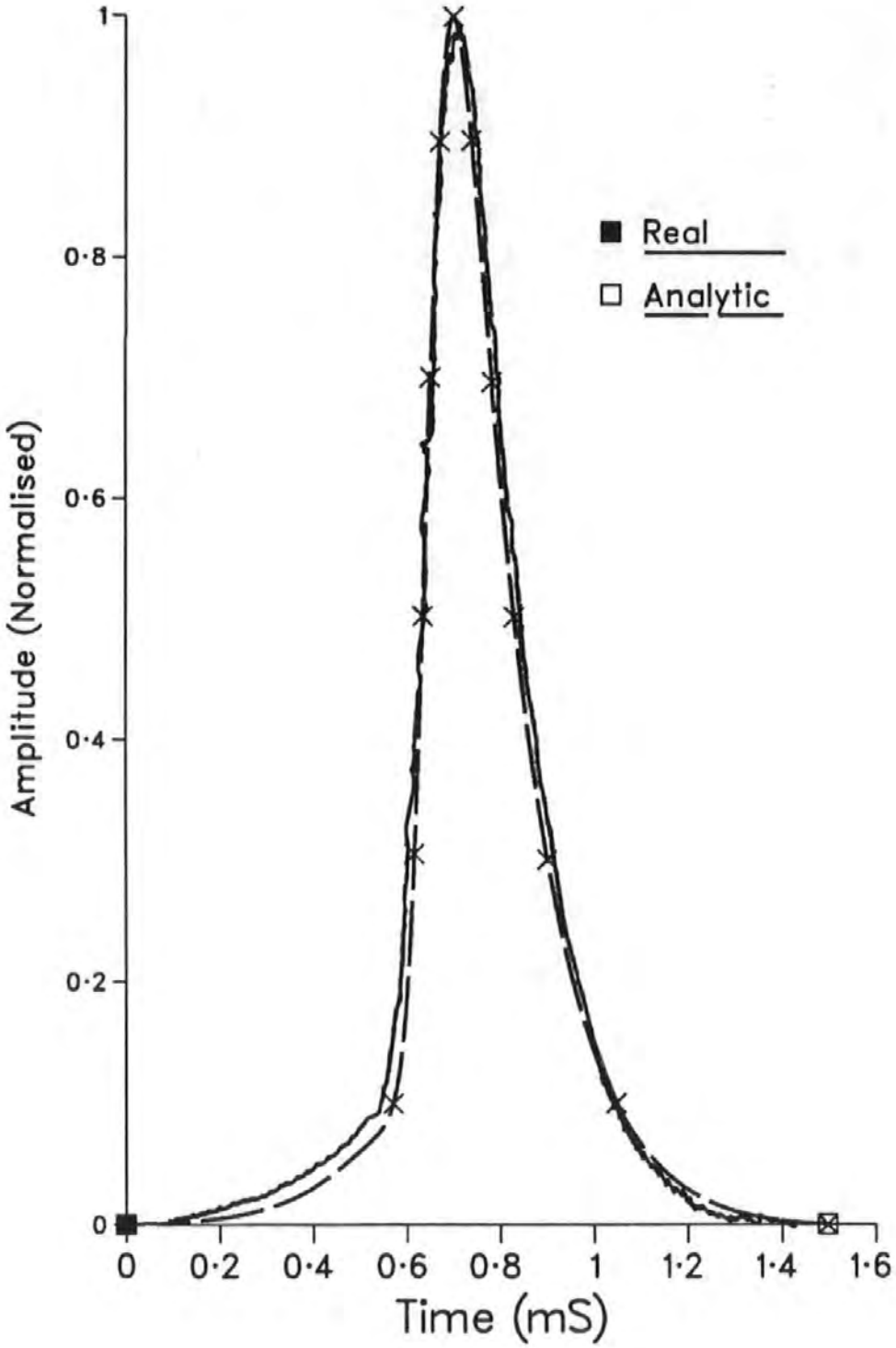


Fig. 4.1. Inductive Head Isolated Pulses.

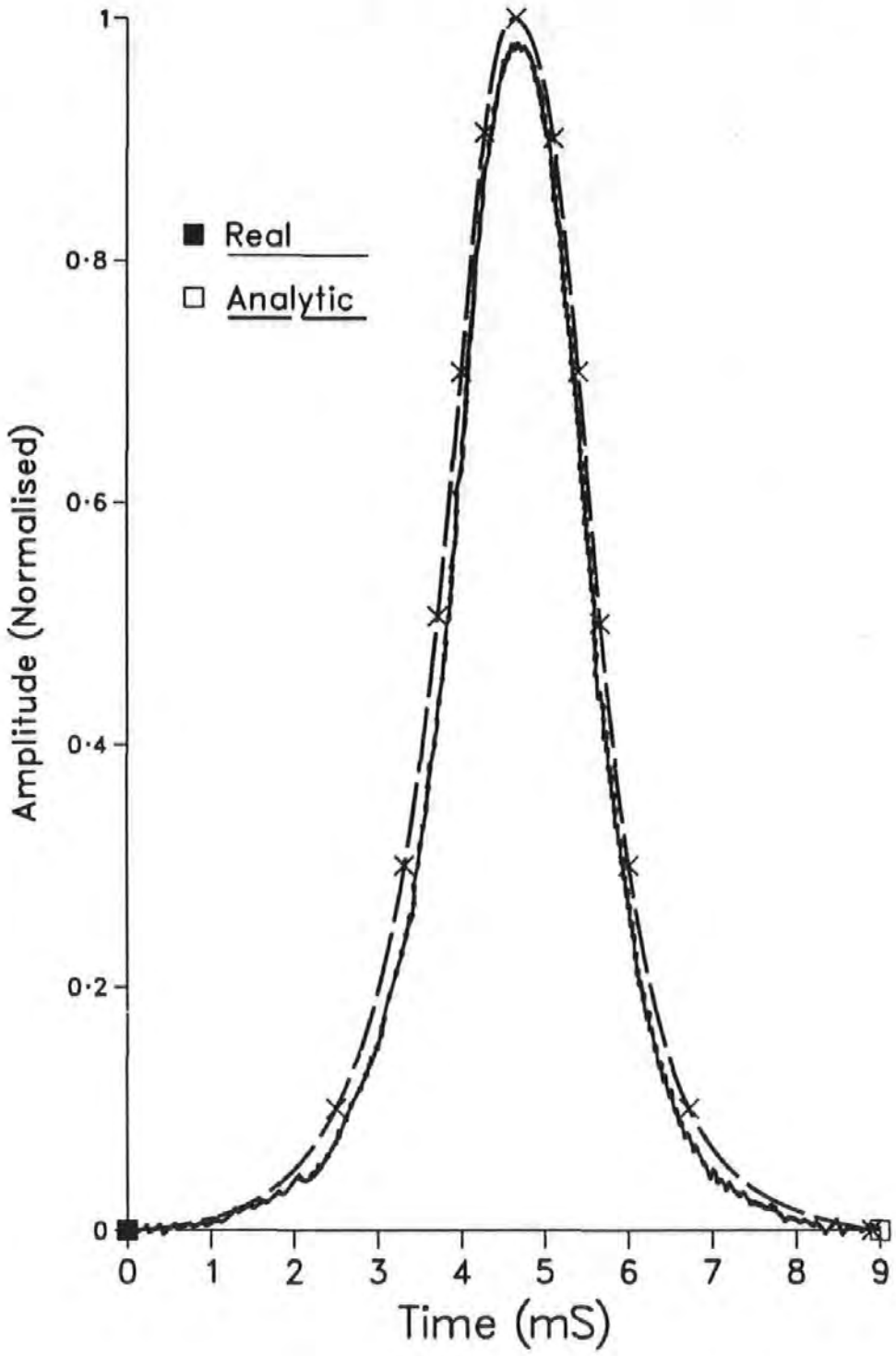


Fig. 4.2. Magneto-Resistive Head Isolated Pulses.

using selected sections from specific pulses were the above results obtained.

ii) There is no theoretical basis to the equations produced. Therefore the pulses cannot be manipulated in any manner that can be related to the recording process. For example, if Lorentzian type equations had been used, it would have been possible to investigate (for example) the effect of write amplifier rise-time by changing the arctangent parameter. The technique therefore sacrifices extensibility for accuracy.

As the analytically derived isolated pulses were accurate representations of the pulses to be simulated, when combined using LPS, the resultant waveforms were similarly accurate, see figure 4.3.

4.3. The Effect of Write Current on Replay Waveform.

The fringing field emanating from the write head gap must be greater than the coercivity of the magnetic medium in order to record information. The magnitude of this field is dependent on the write current. As the Inductive Head was designed for use in an analogue audio system, there was no information regarding DC write currents or the resulting field strengths. An investigation into the effect of write current on replay waveforms was therefore undertaken. Three different types of cassette were investigated:

i) IEC/Type I, a TDK AD 90 (gamma-Ferric Oxide).

ii) IEC/Type II, a TDK SA 90 ('Pseudo-Chrome').

iii) IEC/Type IV, a TDK MA-X 90 (Metal particle).

Each cassette was recorded with a range of write currents, from 0.1mA to 2mA. From each of the replayed waveforms two measurements were taken:

i) Signal Amplitude, (Peak-to-Peak).

ii) Peak-Droop. This is a measure of loss of peak amplitude

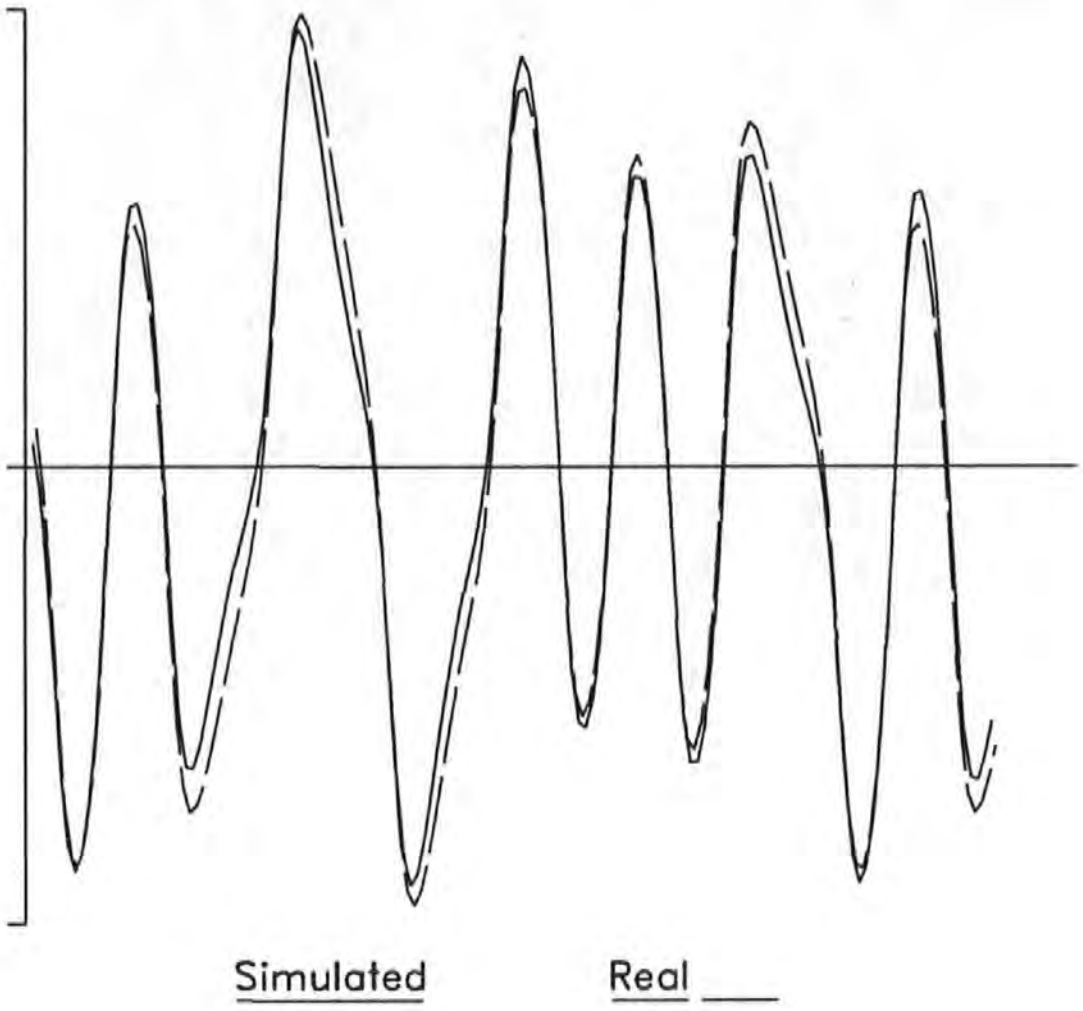


Fig. 4.3. Comparison of Real and Simulated 3125bps Waveforms.

due to Inter-Symbol Interference (ISI), and is illustrated in figure 4.4.

Figure 4.5 illustrates how the Peak-to-Peak amplitude varies with write current for each of the three cassettes. A straightforward explanation can be derived by considering the tape's coercivities and concentrating on the two extremes of recording current.

i) High Record Currents: The greater the coercivity of the tape, the greater the maximum possible recorded magnetic field, the greater the possible rate of change of sensed flux and therefore magnitude of the replayed signal.

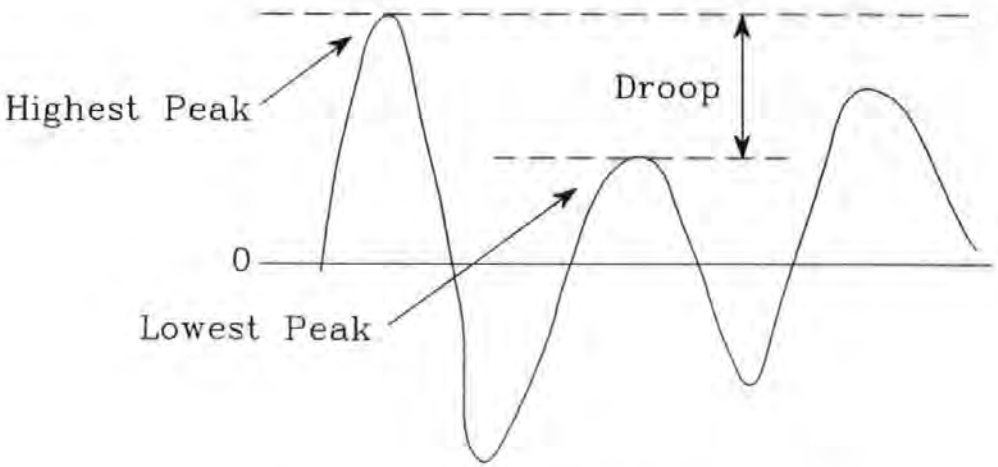


Fig. 4.4. Calculation of Peak-Droop.

ii) Low Record Currents: The smaller the coercivity of the tape, the smaller in magnitude the fringing field needs to be to magnetise the medium, and therefore the smaller the recording current.

Figure 4.6 illustrates how the Peak-Droop varies with recording current for each of the three cassettes. Large values of Droop indicate large amounts of ISI. The higher coercivity tapes therefore exhibit less ISI, suggesting a reduction in PW50. Assuming arctangent transitions, an explanation for this may be gained from equation 4.1 (derived in Appendix B):

$$PW50 = 2/((f_x+d)(f_x+d+\delta) \qquad \text{Equ. 4.1}$$

Although the coating thickness δ and effective spacing d may be lower for the higher coercivity tapes (reducing PW50), a change in the arctangent parameter f_x is the most likely cause. The relationship between the arctangent parameter and the magnetisation (stated in Appendix B) is defined by,

$$M_x \propto \arctan(x/f_x) \qquad \text{Equ. 4.2}$$

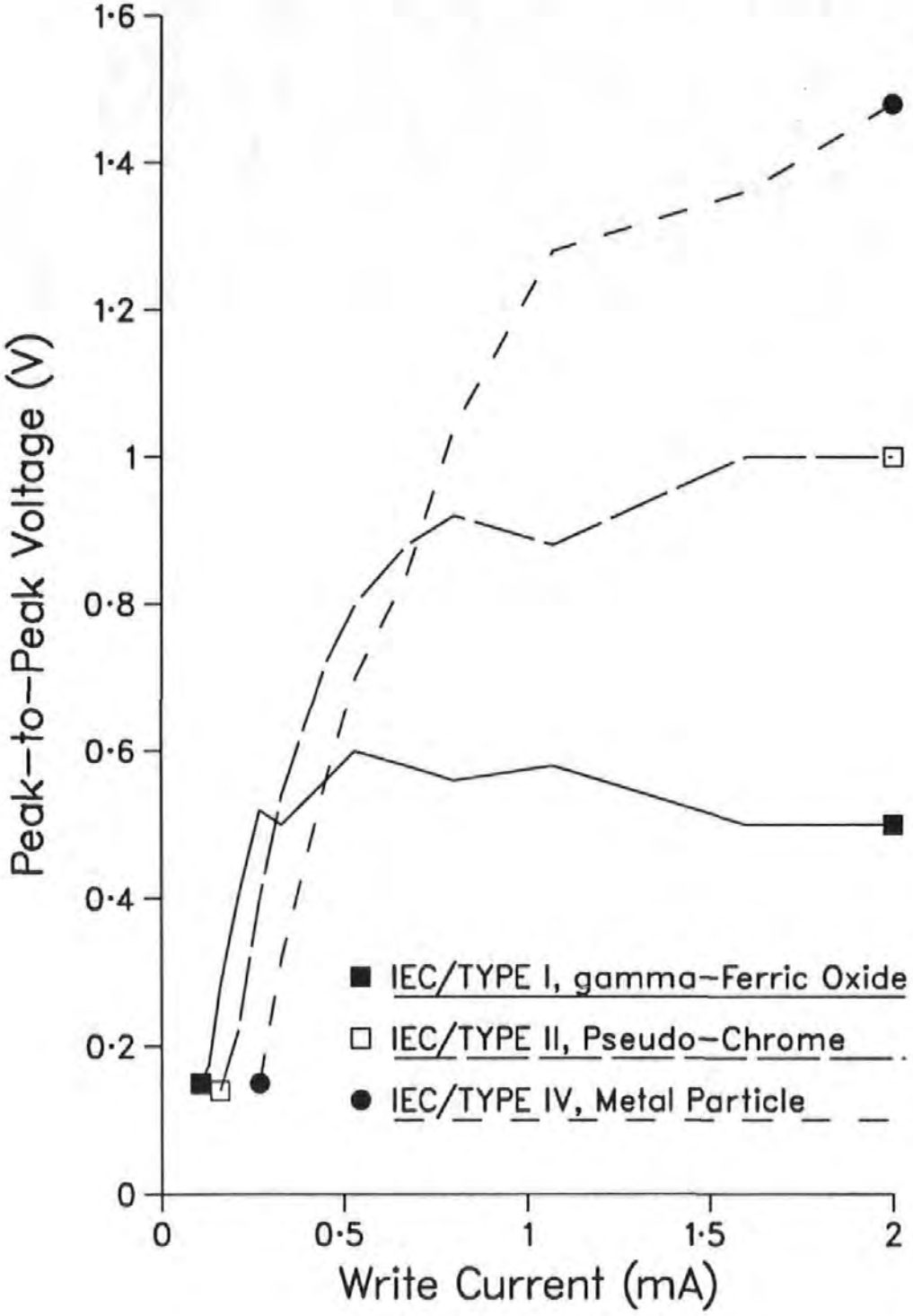


Fig. 4.5. The Effect of Write Current on Peak-to-Peak Amplitude.

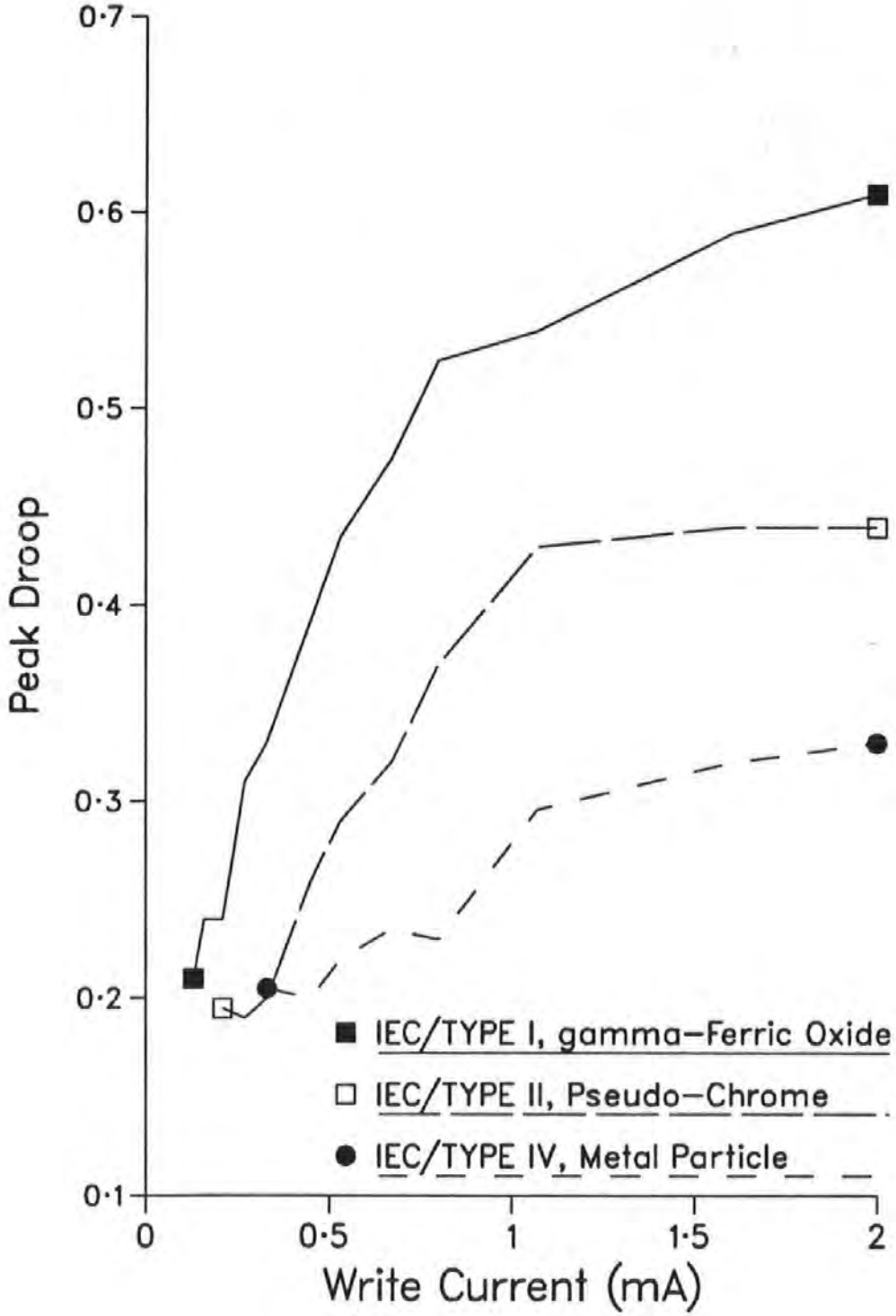


Fig. 4.6. The Effect of Write Current on Peak Droop.

The higher magnetisation results in a reduced arctangent parameter, and is the most likely cause for the reduced PW50 and therefore lower ISI.

Figures 4.5 and 4.6 illustrate the compromise between output amplitude and ISI of the replayed signal. As the replay channel was not equalised, and peak-droop was determined to be the limiting constraint, a write current of 0.33mA was chosen, thereby reducing the peak-droop at the expense of signal amplitude (and therefore SNR).

4.4. Error Rate Profiles.

The following sections detail how various system parameters effected the recording channel's performance, in terms of error rates.

4.4.1. The Effect of Data Rate on Error Rate.

Figure 4.7 illustrates the effect of data rate on the error rate, both for the compact-cassette channel and its simulation. Each data point plotted is the average taken over all 4 tracks. A figure of merit for the system may be calculated from the maximum data rate to error rate ratio. The compact-cassette system has a maximum value of,

$$\frac{4500}{6.75\text{x}10^{-8}} = 6.7\text{x}10^{10}$$

that compares favourably with a figure of $2\text{x}10^{10}$ for a similar compact-cassette system developed by Donnelly (Donnelly, 1989).

The curves are composed of two distinct regions; where the error rate remains fairly constant, and where the error rate increases logarithmically. Errors in the first region were caused primarily by medium related problems or written errors. This was evident from the fact that the errors occurred at the same point during successive replays. The error rate remained fairly constant

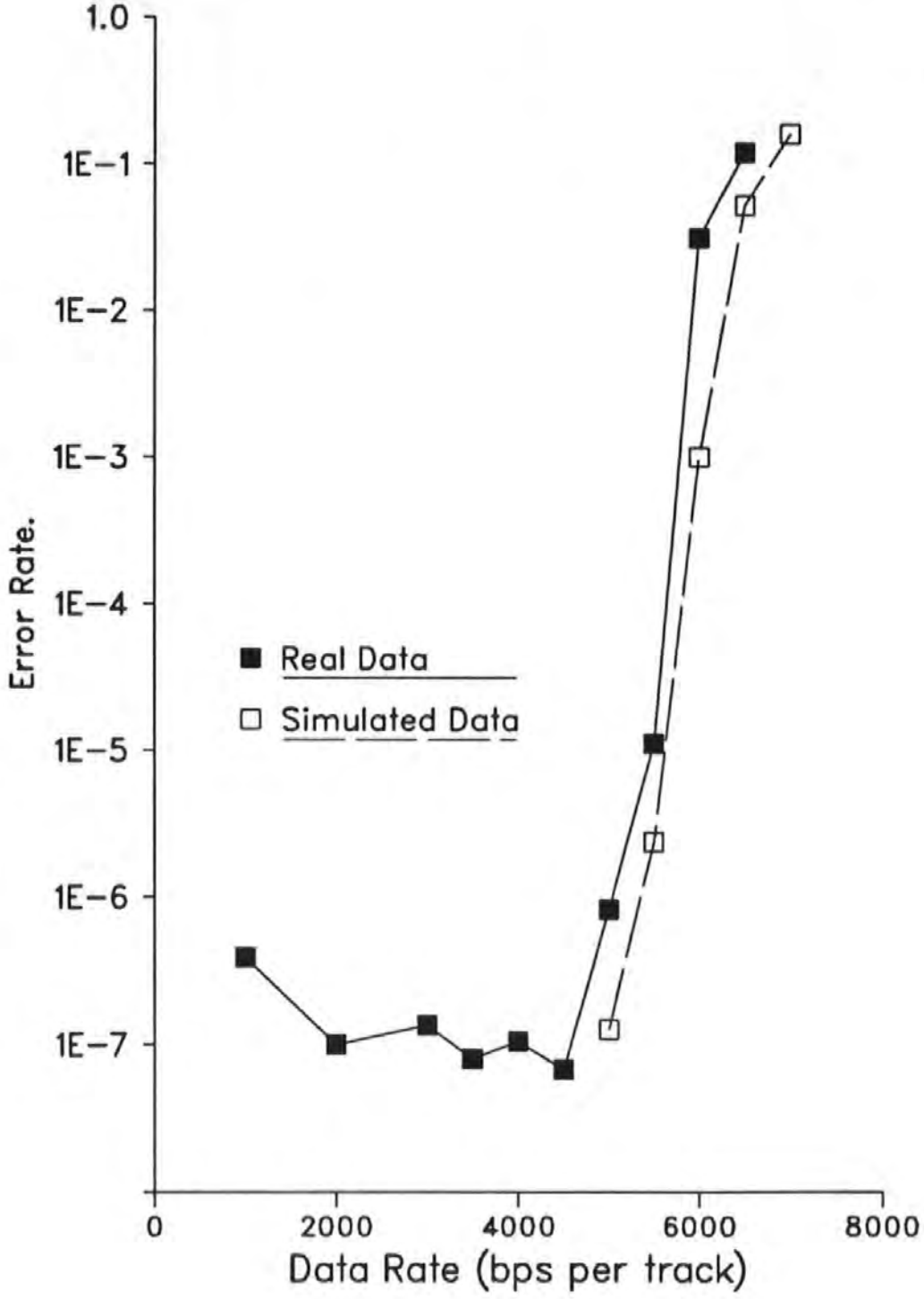


Fig. 4.7. The Effect of Data Rate on Error Rate.

between 1kbps and 4.5kbps, contradicting theory: the error rate would be expected to rise linearly with the data rate as the defective area of magnetic tape affects a greater number of bits. One possible explanation was that the analogue electronics were optimised for operation at 5kbps, and this outweighed the above mechanism.

From around 4500 bits per second, errors start to increase approximately logarithmically. Figure 4.8 shows a typical set of waveforms from the simulation running at 6000 bits per second. Due to ISI, several peaks are significantly reduced in amplitude, some not crossing the 'zero line'. It was at the Polarity Discriminator comparator that these reduced amplitude peaks manifests themselves as errors. The same peaks are successfully converted into the gating

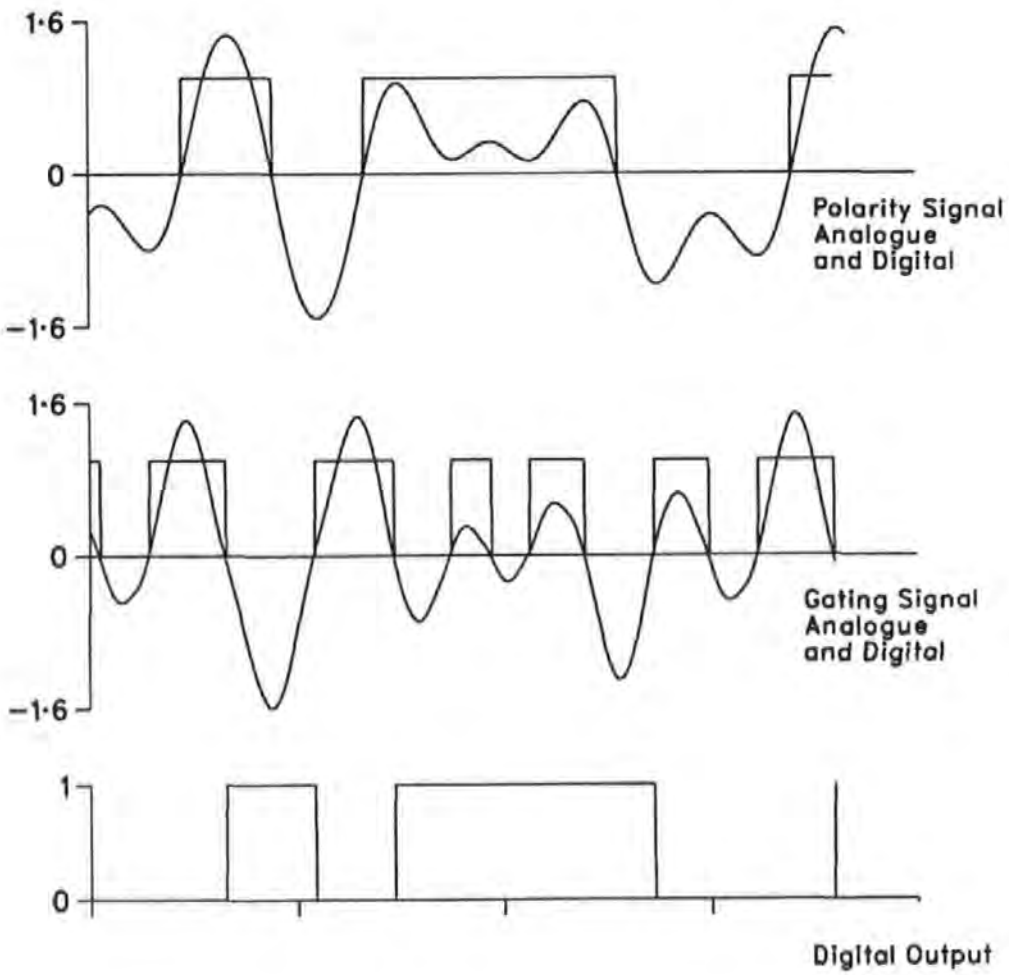


Fig. 4.8. Sample Waveforms for a 6kbps Signal.

signal as they are differentiated, restoring much of the lost amplitude, before input to the gating signal comparator, (differentiation can be used as a simple form of equalisation).

In a noiseless system, the error rate curve past the knee of the characteristic would be a succession of steps. This is because first the worst-case bit pattern fails, followed by the next worst bit pattern, and so on. The combined effect of the noise sources tends to blur or smooth out these steps. These tests were performed with a PRBS of length 7. Without noise, once the critical ISI point for the worst case bit pattern had been exceeded, every one in 7 bits would be in error. The error rate would therefore increase from $1.0x10^{-7}$ to $1.4x10^{-2}$ in one step.

More useful error information would have been elicited had a longer test sequence been used. However long test sequences could not be clearly displayed on the oscilloscope used to monitor signal quality and consequently were not used. Rerunning the tests with longer PRBSs would be straightforward as the model was designed to generate and check sequences up to 32767 bits long. Note however, the longer the PRBS, the lower the probability of filling the reference PRBS register with an error-free sequence for synchronisation. Therefore short test sequences still need to be used when investigating high error rates.

Referring again to figure 4.7, the only significant deviation between the compact-cassette data and the simulated data was a shift of approximately 300bps (or 6% of 5kbps). This suggests the simulated pulse was narrower than the actual pulse. The isolated pulses used to specify the shape of the reference pulse were captured from the first replay of the first recording made on new tape. It was not practical to use new tape for each test, and so the results presented here for the compact-cassette system were obtained from cassettes used many times. When isolated pulses from the most frequently used cassettes were measured, the pulse width was found to be up to 22% wider (0.246mS compared to 0.201mS). This broad range of pulse widths more than covers the deviation between the compact-cassette system and its model. One interpretation of this deviation is the model simulates the compact-cassette system using new cassettes, i.e. 'best-case'

conditions.

Inspecting sections of tape recorded and replayed hundreds of times revealed creases along the length of such tapes. This would cause spacing loss and therefore a broadening of the detected transition. Examining the magnetic head revealed considerable wear, and was the most probable cause of these creases. (The head had by this time been in use for many thousands of hours.)

The error rate curve is affected by nearly all the components of the model, and therefore figure 4.7 was one of the main metrics used to assess the accuracy of the simulation. Based on this premise, the basic model was judged to be an accurate representation of the compact-cassette system.

It should be noted that when the model was simulating a 5kbps data stream, just 5 data bits were simulated per second. Because of this low simulation rate, the model had to be run continuously for weeks for the low error rate measurements to be made. Tests that resulted in error rates of $1x10^{-5}$ or less only detected tens of bits in error. This means that the very low error rates stated carry a very low confidence value. Some models allow very low error rates to be estimated from higher error rates simply from extrapolation (e.g. Katz et al., 1979).

The model was used to simulate the projected performance of the 18-track Magneto-Resistive head. The results are shown in figure 4.9. What is immediately apparent was the reduction, of approximately an order of magnitude, in data rates. There are two main reasons for this poor performance:

i) Incomplete fabrication. As stated in section 2.4.4 the MR head used was a partial fabrication of a more complex design. As a consequence, the pole pieces were not present. Although primarily used in the write process, they had a secondary function as magnetic Shields. Without the pole-pieces, the head was effectively an Un-Shielded MR (UMR) design. To a rough approximation (Jeffers et al., 1984) the PW50 of an UMR element is equal to its height: 40μm in this design. This corresponds to a PW50 of 0.83mS (compared to a measured PW50 of 1.95mS) at a tape speed of $4.75cms^{-1}$.

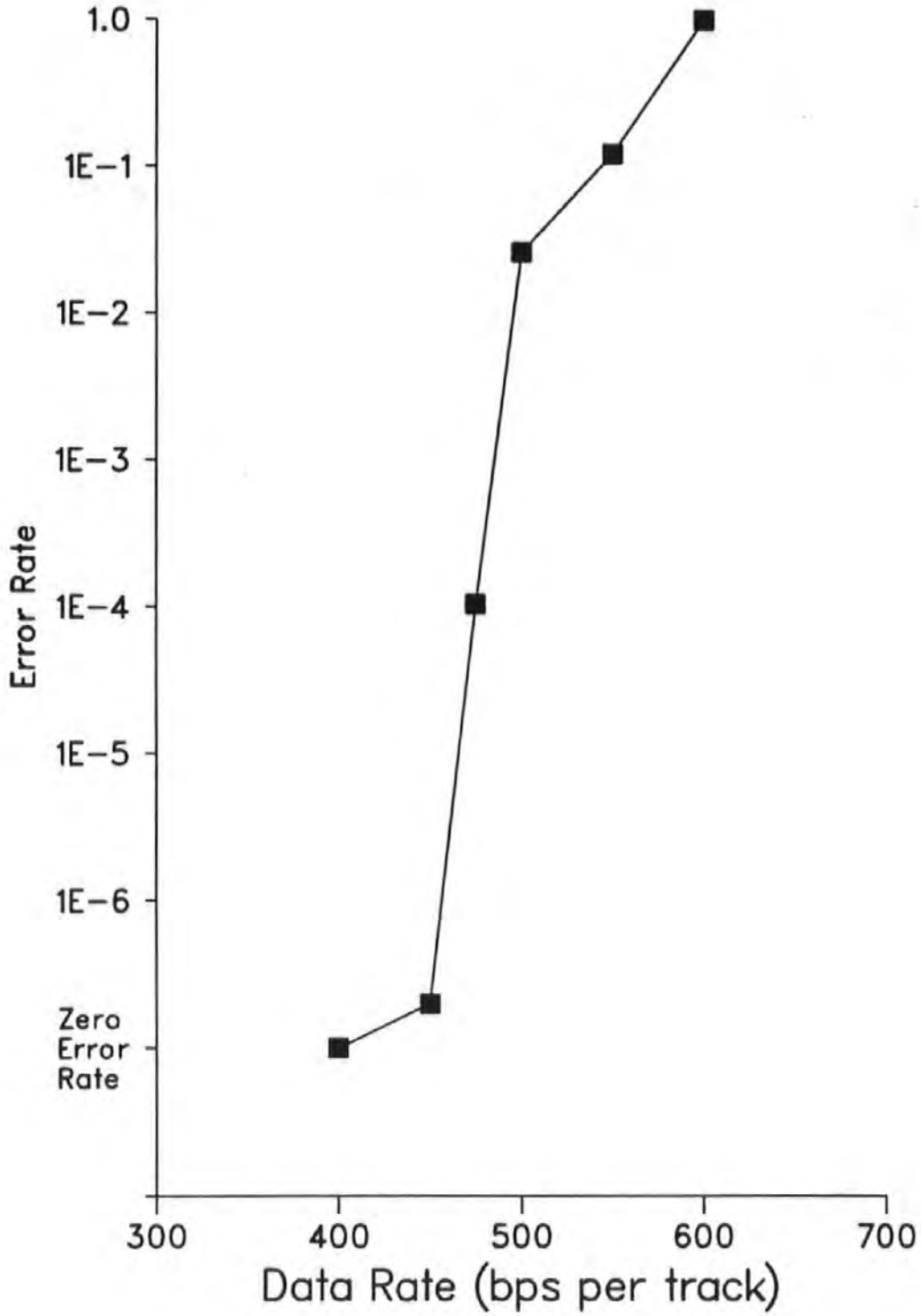


Fig. 4.9. The Simulated MR Head, Data Rate Error Profile.

ii) Fabrication problems. After the MR read elements and lead-outs had been fabricated, the substrate had a covering of glass bonded to it. The 'substrate plus glass' was then bonded between two ceramic cheeks to reduce wear. Problems aligning all these components resulted in the MR elements being slightly recessed. This results in a spacing-loss that widens the PW50, further degrading performance.

Although the MR results are disappointing in performance terms, the ease with which these results were obtained demonstrates the power and flexibility of the model as a development tool.

4.4.2. Simulation of a Peak Detector.

As stated in section 4.4.1, the Peak Polarity signal of the GXO detector fails (due to ISI) at lower data rates than the Gating signal. As the gating signal contains all the information needed to decode the data, the model was used to investigate the performance of the system using a Peak detector which does not use a Polarity signal. Using the differentiator and comparator elements of the GXO detector, a simple Peak detector was assembled, see figure 4.10. The data rate versus error rate profile obtained using the Peak detector is shown in figure 4.11 together with that for the simulated GXO detector system.

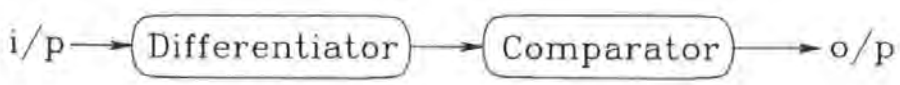


Fig. 4.10. ***occam*** Process Diagram of Peak Detector.

The modification to the detector involved removing three lines of code and changing an ***occam*** channel name. This modification was the software equivalent of removing 3 integrated circuits from the hardware GXO detector and changing a wire to route the signal past the now redundant circuitry. These results are not presented here to extol this 'trial and error' design methodology. They are presented

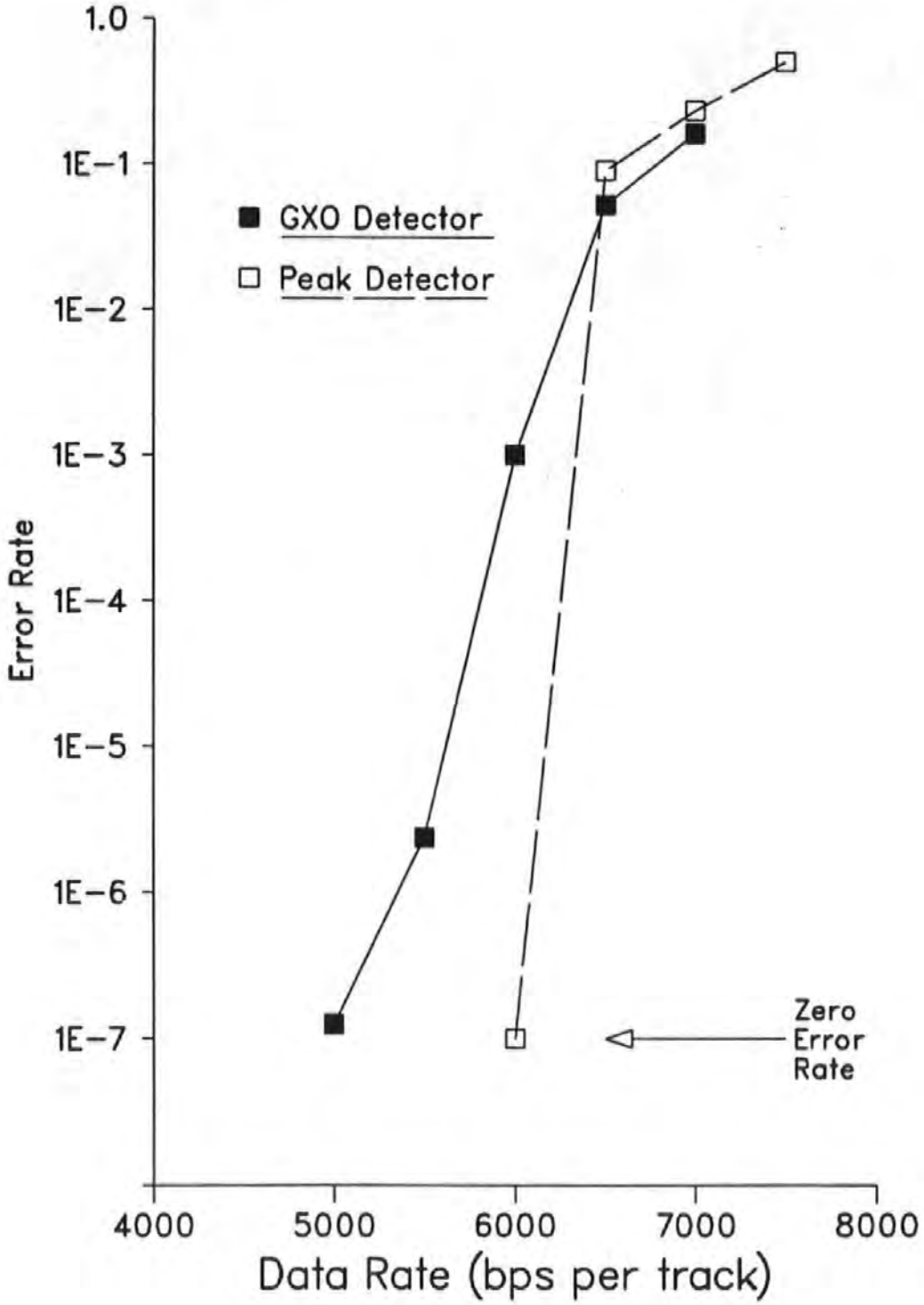


Fig. 4.11. Comparison of Simulated Peak and GXO Detector Performance.

to show the flexibility of the model and how well suited *occam* was to this style of modelling.

Figure 4.11 shows the improved performance of the Peak detector compared to that of the GXO detector. At low error rates the capacity of the channel was increased by approximately 20%. Whereas it was peak-attenuation (caused by ISI) that produced errors when the GXO was used, it was the peak-shift that produced the errors when the Peak detector was used. As the data rate increases, the amount of peak-shift increases until the period between transitions is in error by more than half a code bit. This will cause the decoder to incorrectly decode the sequence.

4.4.3. Variation of Error Rate Profiles between Replays.

The error checking process can be used to record the position of errors on the tape. In order to investigate how the positional error rate profile varies the following tests were performed:

i) Cassette AC Bulk erased.

ii) Cassette recorded (with 5.5kbps Bi-Phase-L encoded PRBS).

iii) Cassette replayed and rewound 4 times (storing the positional error rate profiles as Tests 1 to 4).

iv) Steps i), ii), and iii) repeated with the same cassette (storing the positional error rate profiles as Tests 5 to 8).

The higher than normal data rate ensured a significant number of errors would be detected. Figure 4.12 shows all 8 error rate profiles. From a visual inspection the test profiles form into two groups: before and after the re-recording (i.e. between Test 4 and Test 5). To produce a more quantifiable interpretation, the 8 profiles were cross-correlated with one another. The results are shown in Table 4.1. These figures confirm the visual observation that the positional error rate profiles are highly correlated between tests performed on the same recording, and poorly correlated between tests performed on different recordings.

Figure 4.13 plots the correlation coefficients against the number of tests between correlations. The obvious inverse

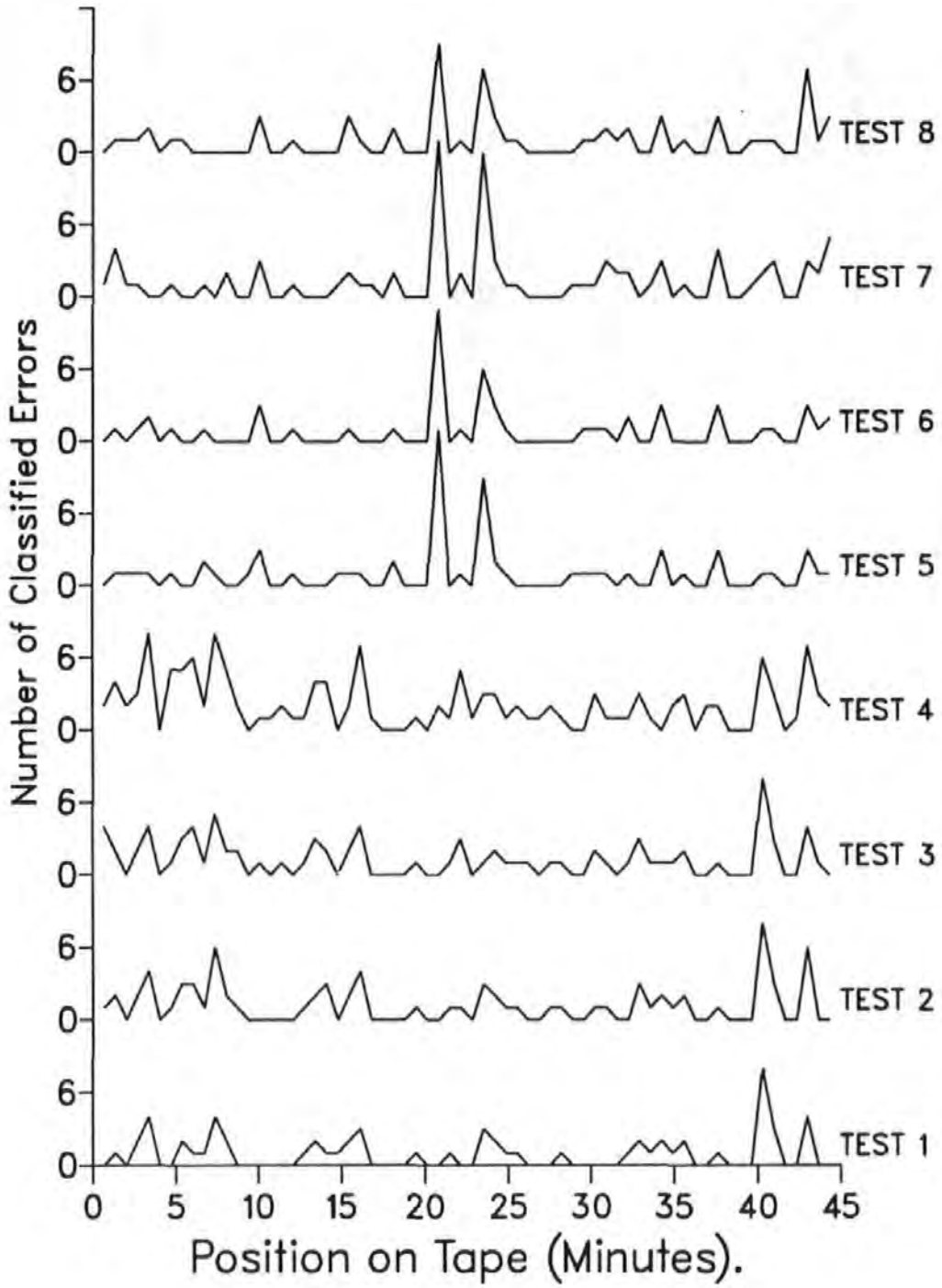


Fig. 4.12. Error Distributions for Eight Successive Replays.

Test	1	2	3	4	5	6	7
2	0.930						
3	0.819	0.905					
4	0.678	0.816	0.825				
5	0.145	0.105	-0.02	0.097			
6	0.148	0.102	0.005	0.116	0.959		
7	0.146	0.098	-0.01	0.094	0.920	0.906	
8	0.247	0.228	0.077	0.205	0.878	0.915	0.874

Table 4.1. Cross-Correlation Coefficients between Successive Replays.

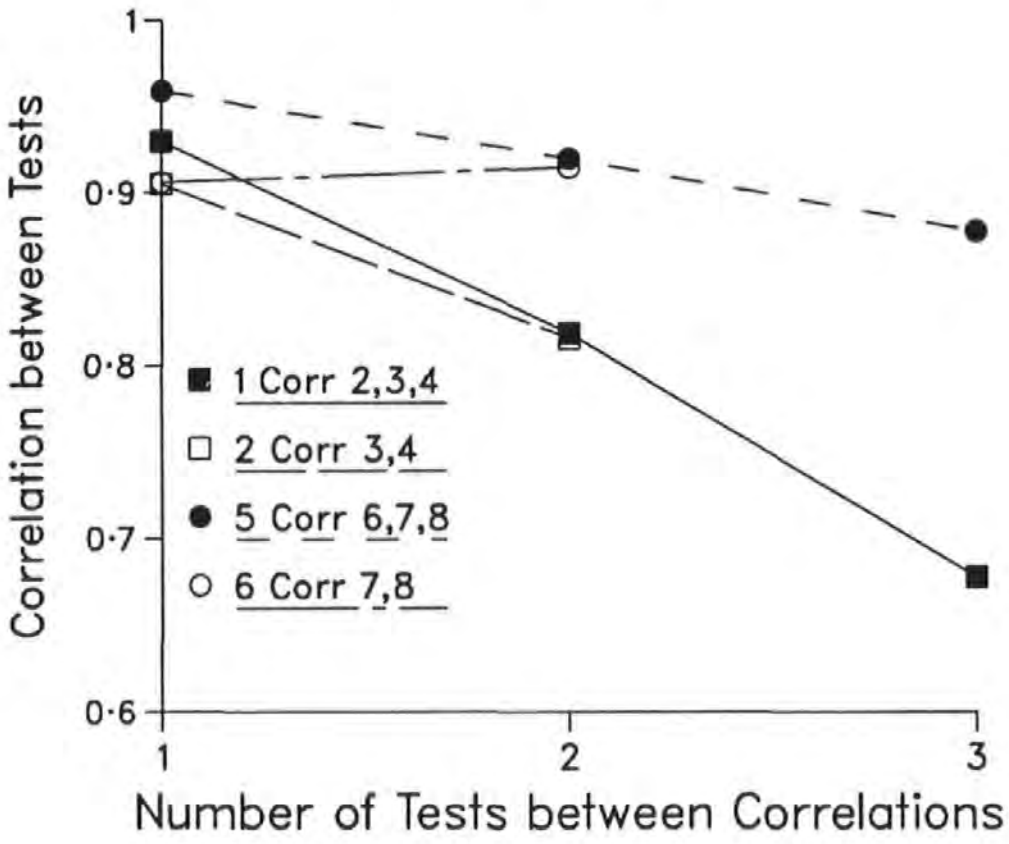


Fig. 4.13. The Degradation of Correlation Coefficients between Successive Replays.

proportionality strongly suggests a non-random process was causing the error rate profiles to vary. Examining the error rates for successive tests revealed a slight increase. One suggested cause for such a trend is loss of magnetic medium due to wear.

These results discouraged further investigation into using a knowledge of the cassettes positional error profile from one recording, to determine the error correction scheme applied during successive recordings (for example dedicating a higher number of bits to error correction for parts of the tape that had a poor error performance on the last recording). It does however suggest a write/read/write head arrangement may enhance performance: the data being verified immediately after being written. The following write head could then either re-write the data or append specially tailored error correction data to allow for correct decoding during subsequent replays.

4.4.4. Head Azimuth Skew.

Azimuth skew can cause errors in a multiple-track system in three ways:

i) By causing misalignment of data with respect to adjacent tracks.

ii) By attenuating the replay signal.

iii) By causing peak-shift.

Figure 4.14 shows the error rate profile caused by azimuth skew. The azimuth skew can be seen to have had little effect on the error rate until approximately 20 minutes of arc. In terms of misaligned data this corresponds to 1.9 data bits of misalignment between the midpoints of tracks 1 and 4. In a parallel sampling system, data skew greater than 1/4 of a data bit would cause errors. This result verifies that treating each track separately removes the problem of data skew between tracks.

In terms of signal attenuation, from equation 4.3 (see section 3.3.2.3), 20 minutes of arc corresponds to an attenuation of 2.0dB at

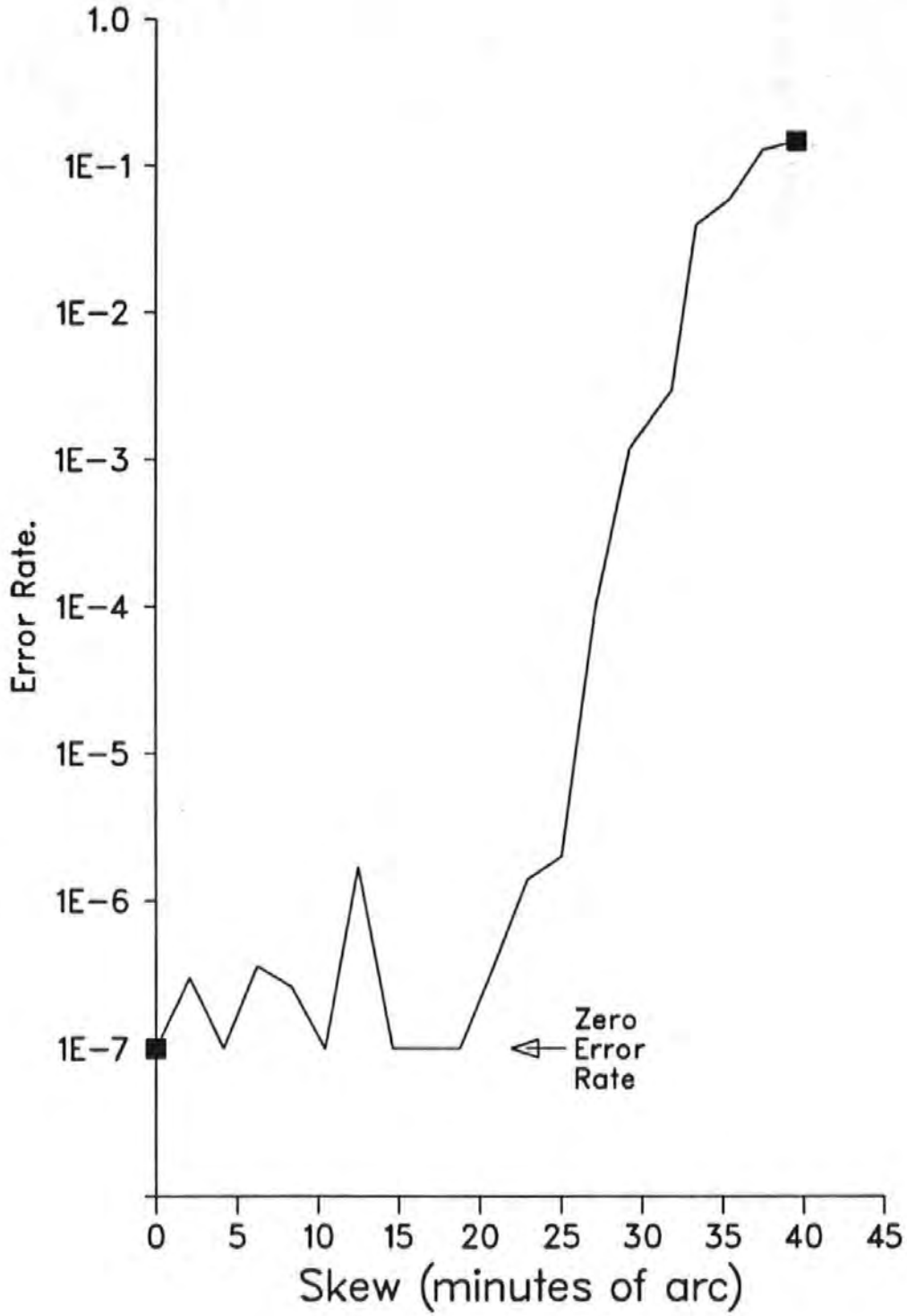


Fig. 4.14. The Effect of Azimuth Skew on Error Rate.

5kHz (assuming a track width of 0.598mm and tape speed of 4.75cm.s^{-1}). This amount of azimuth skew was considerably in excess of the 4.4 minutes of arc encountered under normal operating conditions (Donnelly, 1989).

$$20.\log_{10}\left[\frac{\sin(kw\theta/2)}{(kw\theta/2)}\right] \qquad \text{Equ. 4.3}$$

The impact of the resultant signal attenuation is largely dependant on packing density. At low packing densities, the peak-attenuation caused by ISI is less. This allows higher levels of azimuth skew to be introduced before the combined attenuation is sufficient to cause errors. One of the few benefits of reducing track widths is that azimuth skew attenuates the replayed signals less. A 50μm wide read head (i.e. the width of the MR read element detailed in section 2.4.4) would suffer just 0.01dB of attenuation reading a 5kHz signal when skewed by 20 minutes of arc.

As azimuth skew modifies the phase as well as the frequency response of the channel, the location of peaks will be altered or shifted. In a peak-shift limited system this may cause errors. The compact-cassette system was peak-attenuation limited, therefore the effects of peak-shift caused by azimuth skew were insignificant.

Thus:

i) Systems can be designed to be unaffected by misalignment of data between tracks.

ii) The effect of azimuth skew on the error rate may be offset by reducing the packing density.

iii) The effect of azimuth skew reduces with track width.

Azimuth skew therefore was not viewed as a severe limitation on the ultimate performance of digital magnetic recorders.

4.4.5. The Effect of Lateral Head Displacement.

Section 2.4.2 detailed the modifications made to the compact-cassette head mounting arrangements to allow the head to be displaced from its optimal position, whilst section 3.3.2.5 detailed how this

displacement was simulated. This section presents the results of the Lateral Head Displacement (LHD) investigation. (To demonstrate the level of detail the error classification scheme provides, the complete set of classified error results are presented for the compact-cassette system).

Figures 4.15 to 4.22 show the effect of LHD on error rates (as categorised in section 3.2.2.4) for each track. Figures 4.15 and 4.16 show the overall error rate, figures 4.17 to 4.20 the error rates for the individual burst error lengths for each track, and figures 4.21 and 4.22 the loss of synchronisation. The direction of LHD introduced resulted in the n^{th} track moving toward the $(n-1)^{th}$ track, with the 1^{st} track moving towards the edge of the tape. Due to the compact-cassette track format, this results in tracks 2 and 4 reading similarly distorted signals, as do tracks 1 and 3 (for values of LHD less than the track separation). The large increase in errors for track 2 at 0.2mm, with no corresponding increase in track 4 was attributed to a longitudinal crease observed running the length of the tape, and will not be considered further.

What was immediately apparent from all 8 graphs were the rapid increases and decreases in error rate for small changes in displacement. For displacements up to a critical value there is little increase in error rate. Single-bit errors are the most common in this region. At a critical displacement, the error rate increases rapidly. As the head is displaced further, the n^{th} head starts to be influenced more by the data of the $(n-1)^{th}$ track. A second critical displacement is reached and the error rate for the n^{th} head decreases rapidly as it now starts to read and successfully decode the $(n-1)^{th}$ track. Figure 4.23 shows a simplified representation of the LHD versus error rate profile, where the four critical displacements are indicated by m_{c1} to m_{c4}.

A new figure of merit is proposed, one that specifies the percentage of displacements from which an acceptable level of performance may be achieved. It is important to note that the acceptable level of performance for a head may be achieved reading any track. For this and subsequent results the acceptable level of performance was an error rate of $1x10^{-5}$. The figures of merit can

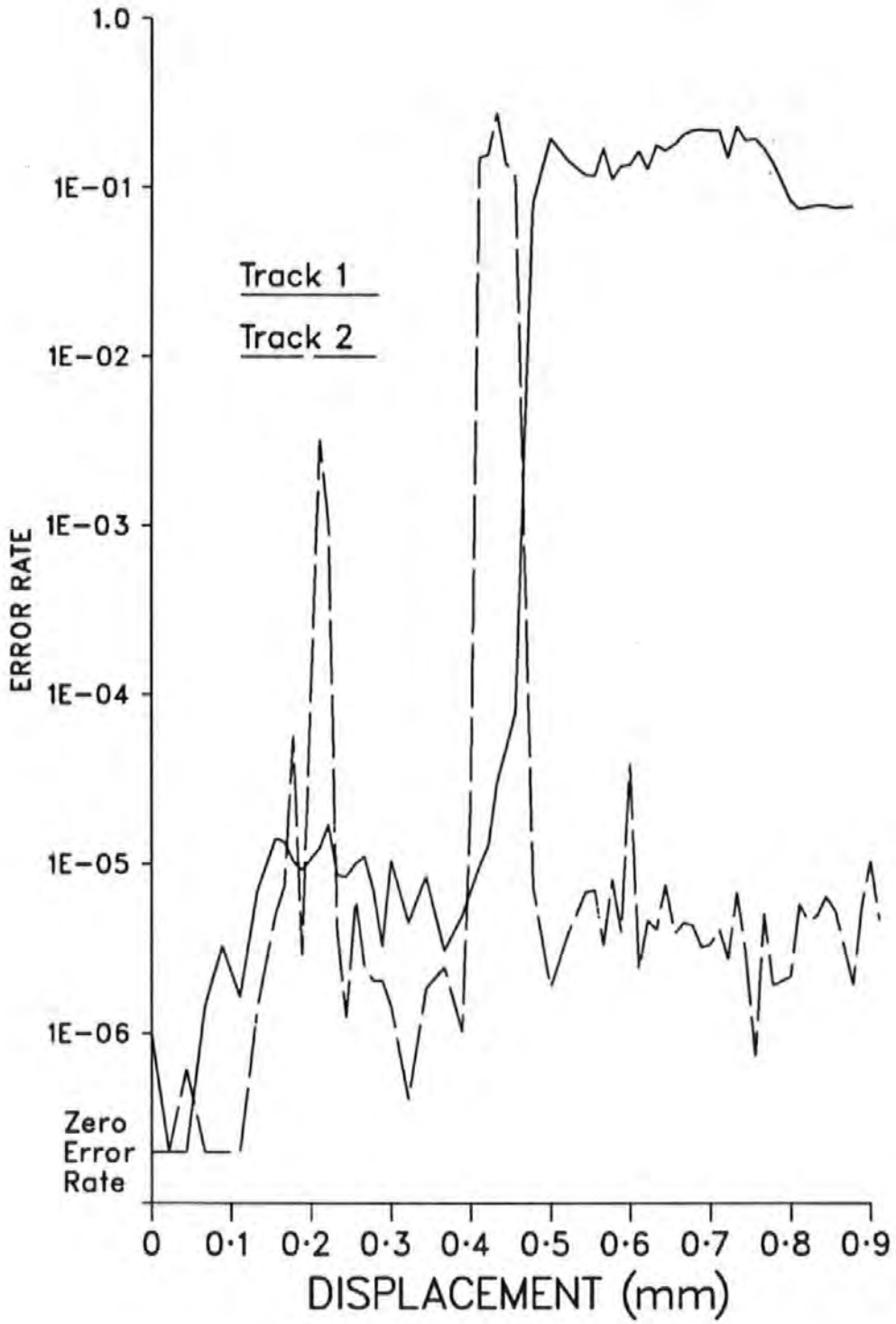


Fig. 4.15. The Effect of Lateral Head Displacement.
on Error Rate (Tracks 1 & 2).

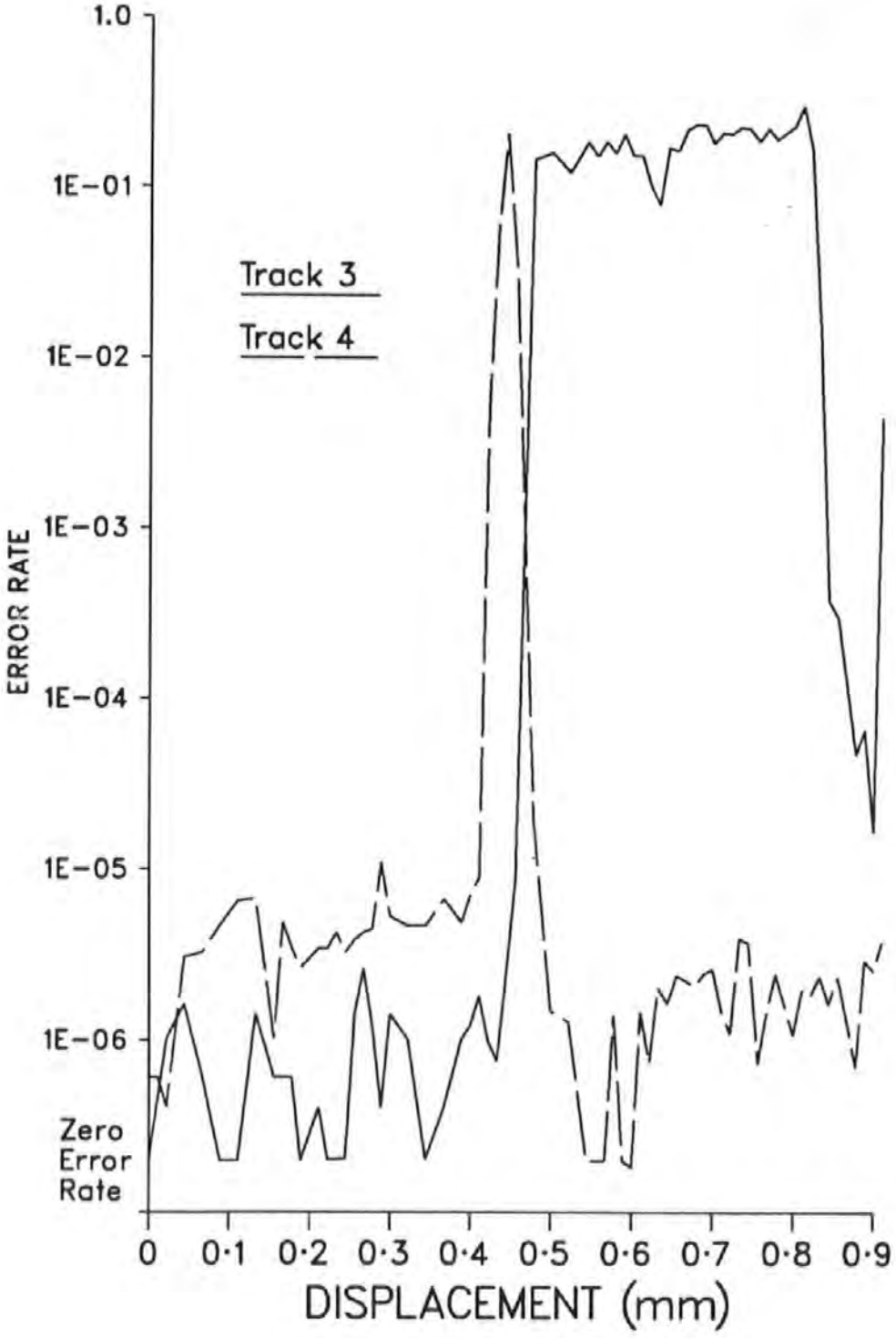


Fig. 4.16. The Effect of Lateral Head Displacement on Error Rate (Tracks 3 & 4).

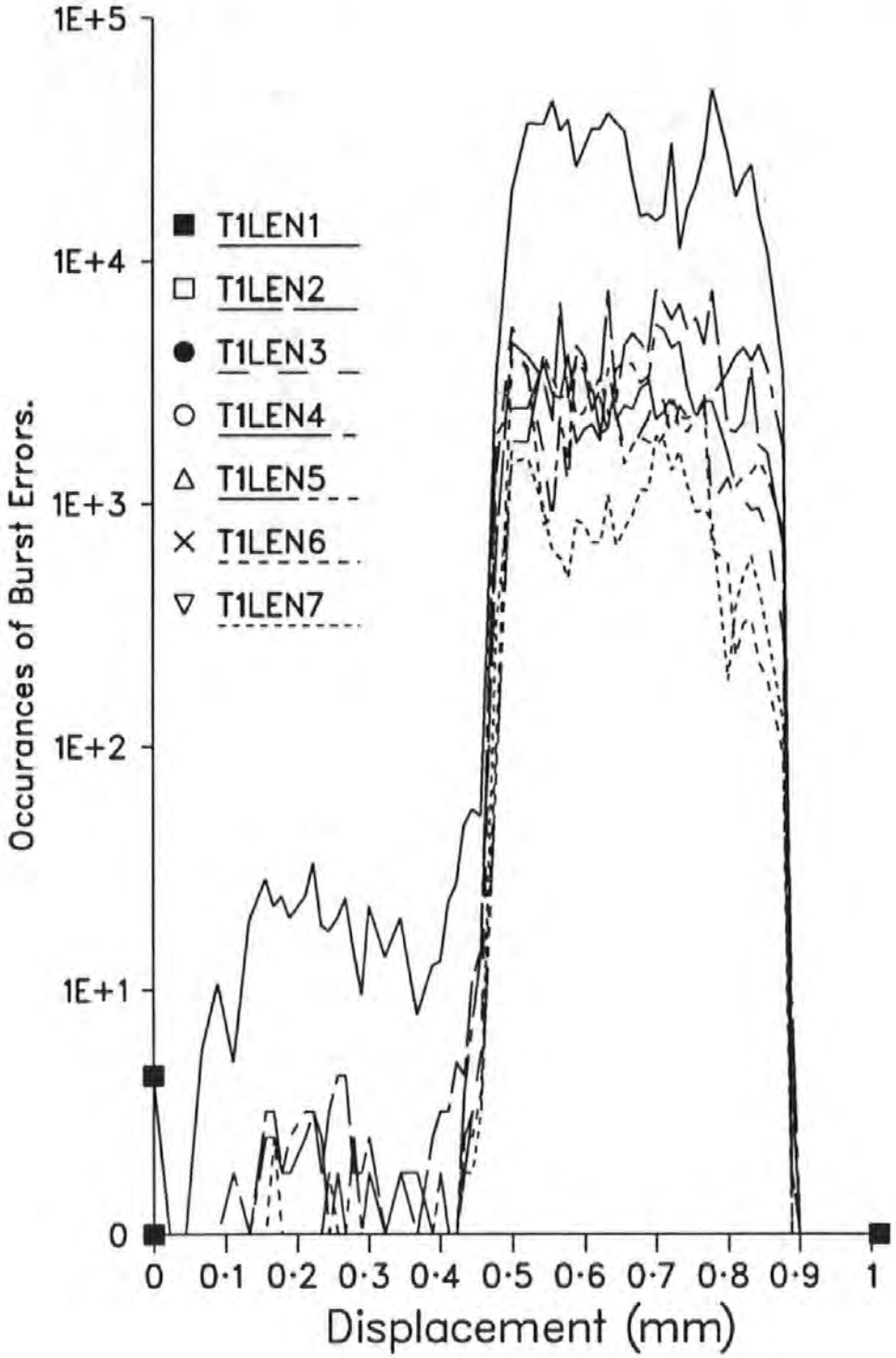


Fig. 4.17. The Effect of Lateral Head Displacement on Error Burst Length of Track 1.

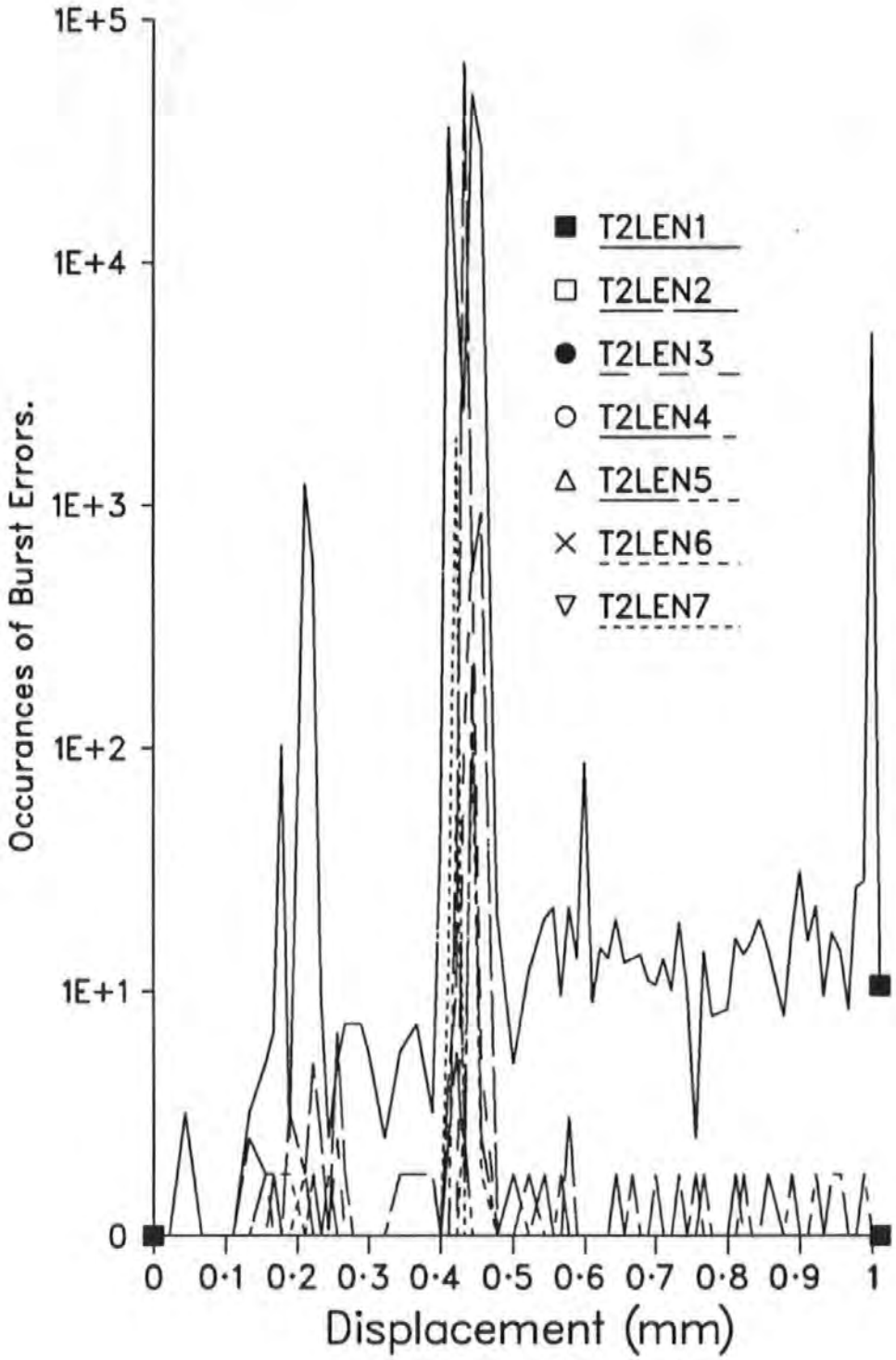


Fig. 4.18. The Effect of Lateral Head Displacement on Error Burst Length of Track 2.

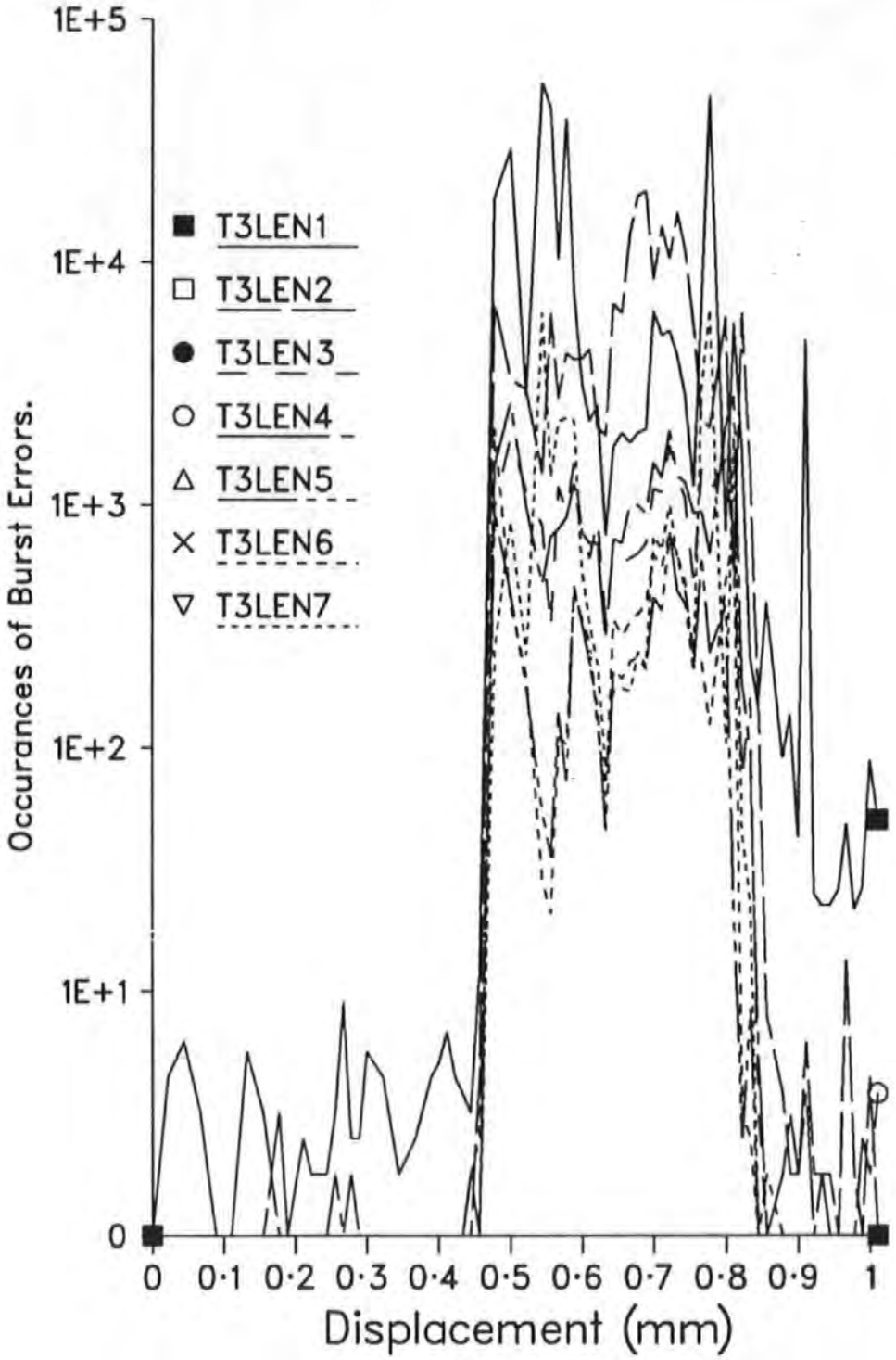


Fig. 4.19. The Effect of Lateral Head Displacement on Error Burst Length of Track 3.

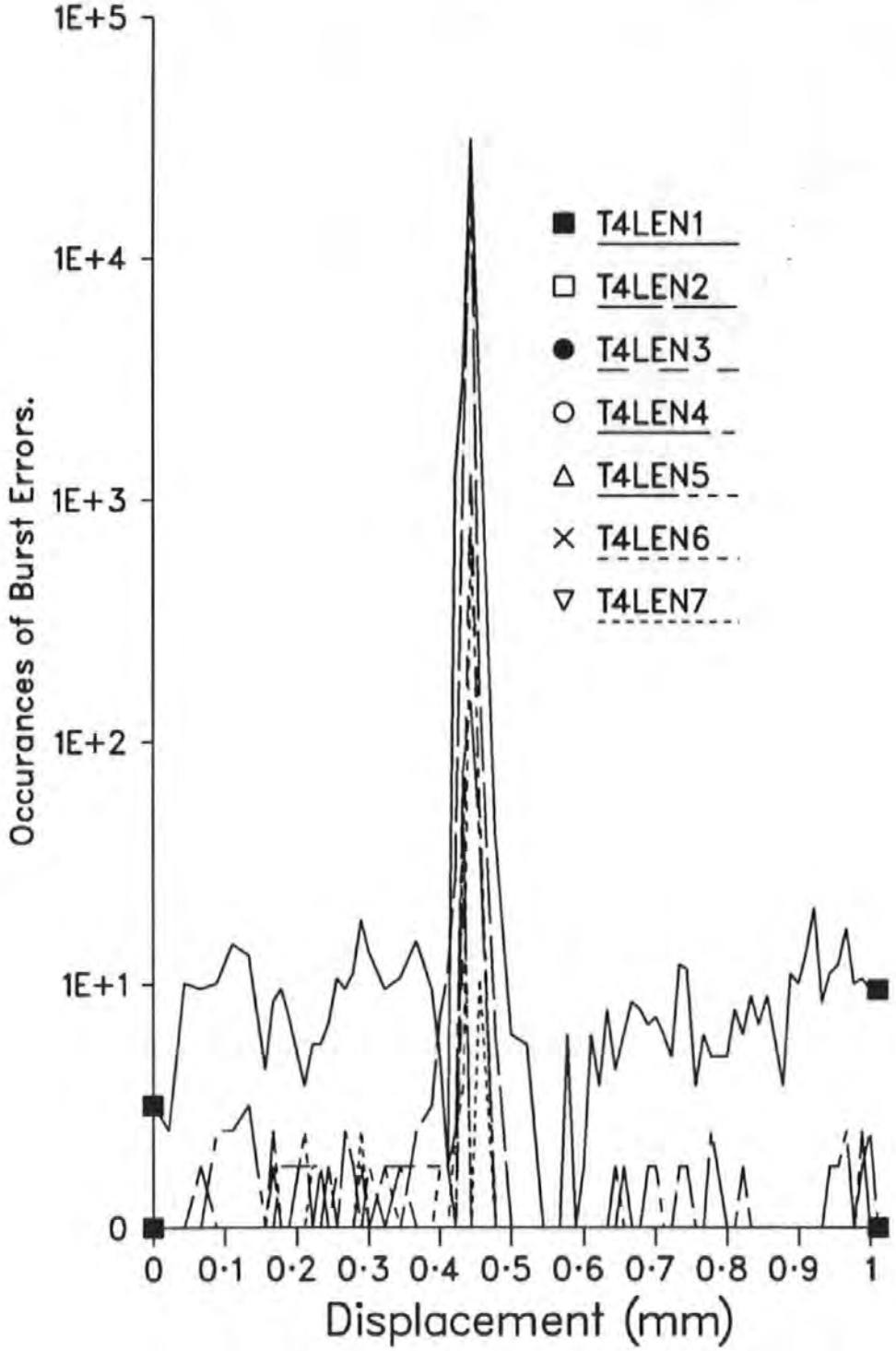


Fig. 4.20. The Effect of Lateral Head Displacement on Error Burst Length of Track 4.

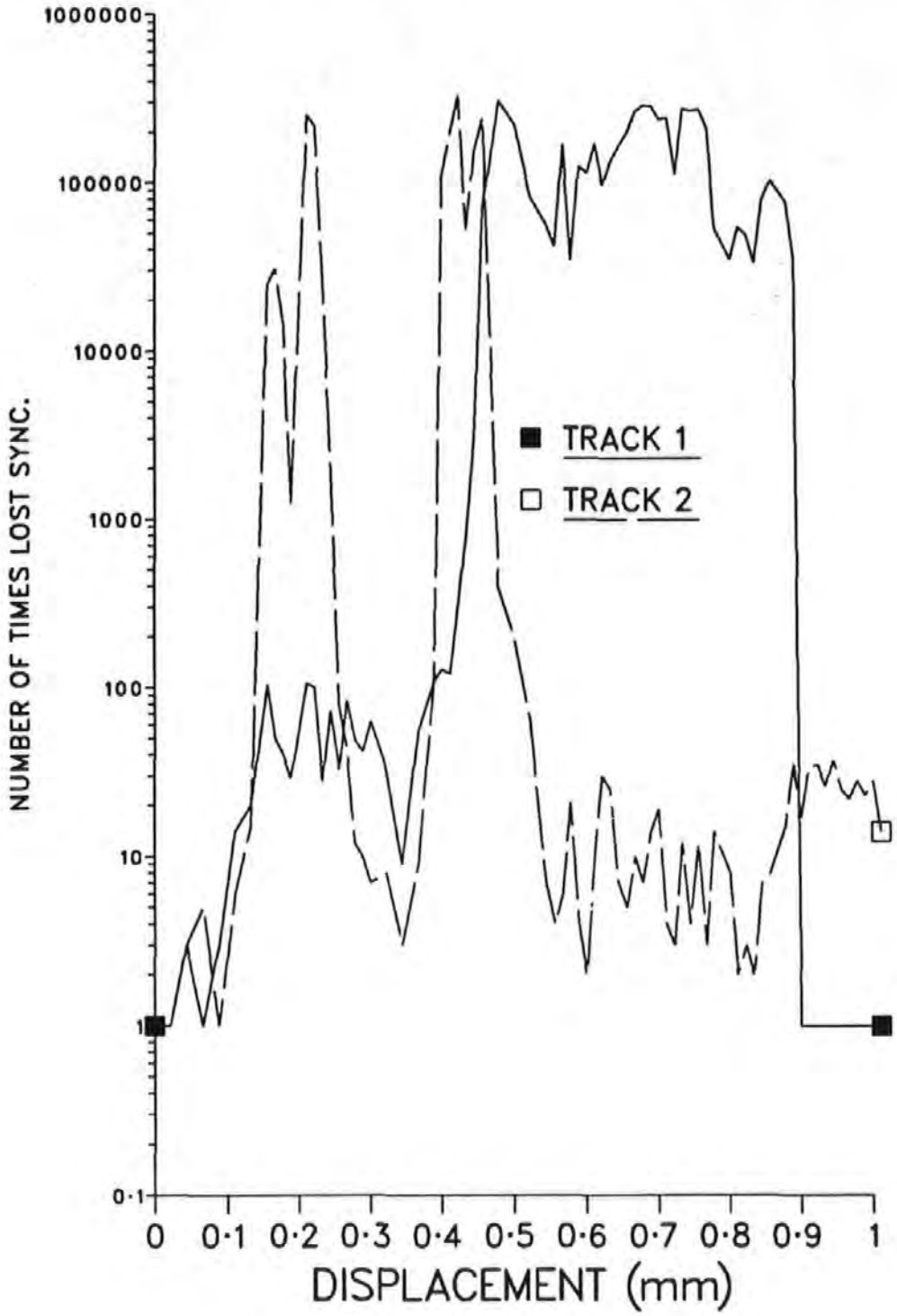


Fig. 4.21. The Effect of Lateral Head Displacement on Loss of Synchronisation (Tracks 1 & 2).

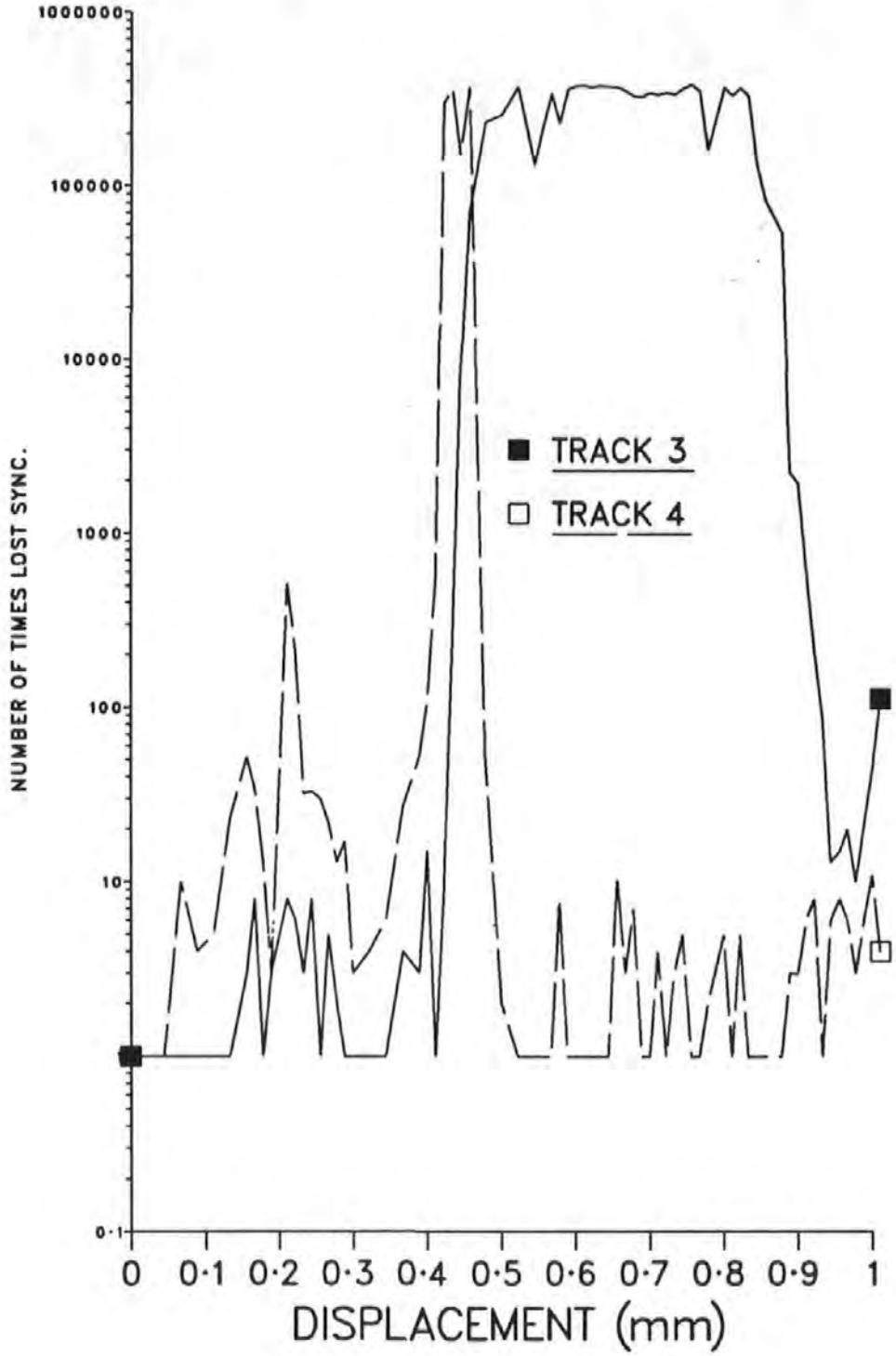


Fig. 4.22. The Effect of Lateral Head Displacement on Loss of Synchronisation (Tracks 3 & 4).

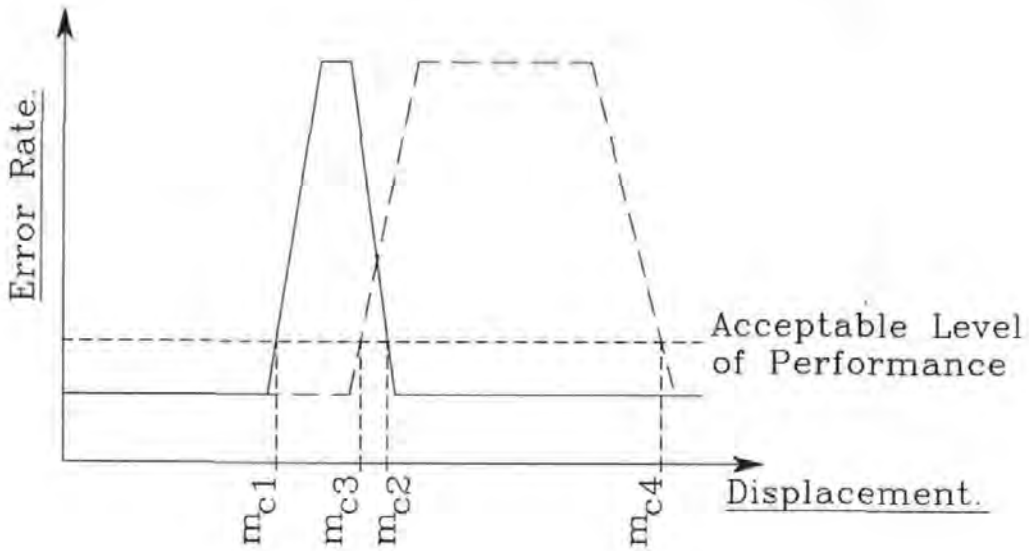


Fig. 4.23. Simplified Error Rate Profiles for LHD.

be calculated for the compact-cassette system from the four critical displacements, i.e.,

$$\left(1 - \frac{m_{c2} - m_{c1}}{m_{c1} + m_{c2}} \right) \quad x\ 100 = 91.2\%$$

$$\left(1 - \frac{m_{c4} - m_{c3}}{m_{c3} + m_{c4}} \right) \quad x\ 100 = 70.0\%$$

The two figures reflect the two different track separations of the compact-cassette system. These results show that the large guard-band between tracks 2 and 3 (designed to reduce cross-talk in the original compact-cassette specification) reduces the range of displacements over which an acceptable level of performance may be achieved.

The three effective track separations may also be calculated from these four critical displacements. From figure 4.24 it can be seen that the ratio $a_{n:n}$ to $a_{n:n-1}$ at m_{c1} is the same as the ratio $a_{n:n-1}$ to $a_{n:n}$ at m_{c2} (a similar argument may be applied to m_{c3} and m_{c4}). Therefore, using $s_{n,n-1}$ to denote the separation between the n^{th} and $(n-1)^{th}$ tracks,

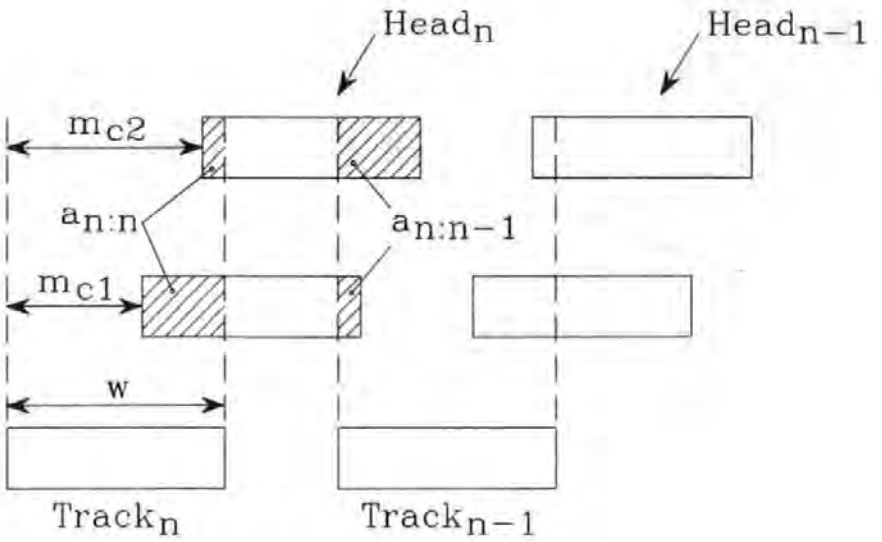


Fig. 4.24. Head and Track arrangements for two complementary critical displacements.

$$s_{2,1} = s_{4,3} = m_{c1} + m_{c2} = 0.405 + 0.483 = 0.888\text{mm}$$
$$s_{3,2} = m_{c3} + m_{c4} = 0.457 + 0.847 = 1.304\text{mm}$$

These figures are compared in Table 4.2 with those for the compact-cassette standard, and those obtained directly from the recording head using a microscope and calibrated graticule.

	Compact-Cassette Standard	Calculated from m_{c1} to m_{c4}	Optical Measurement
$s_{1,2}$ and $s_{3,4}$	0.935	0.888	0.828
$s_{2,3}$	1.290	1.304	1.235

Table 4.2. Comparison of Track Separations.
(Dimensions in mm)

The effect of the interfering signal from the adjacent track can now be quantified. If f(n)=-f(n-1) then the amount of the n^{th} track that the n^{th} head effectively links with is reduced from $a_{n:n}$ to $(a_{n:n}-a_{n:n-1})$. In the general case the Effective Read

Width of the n^{th} head may be defined by:

$$ERW_n = a_{n:n} + ßa_{n:n-1} \qquad \text{Equ. 4.4}$$

where ß is a function introduced to account for the 'mutual interference' between the two track signals. For tracks 2 and 4,

$$a_{n:n} = (w - m_{c1}) \qquad \text{and} \qquad a_{n:n-1} = (w - m_{c2})$$

and for tracks 1 and 3,

$$a_{n:n} = (w - m_{c3}) \qquad \text{and} \qquad a_{n:n-1} = 0$$

(as the track separation is greater than the track width). Assuming the ERW is the same for all tracks, an estimate of the value of ß for this set of tests can be made. For tracks 1 and 3 at m_{c3}, w is less than the track separation, i.e. $a_{n:n-1} = 0$. Therefore,

$$ERW = (w - m_{c1}) + ß(w - m_{c2}) = (w - m_{c3}) \qquad \text{Equ. 4.5}$$

which may be rearranged as,

$$ß = \frac{(m_{c1} - m_{c3})}{(w - m_{c2})} \qquad \text{Equ. 4.6}$$

The track width, w, may be determined experimentally. On a plot of signal amplitude versus displacement, the signal will decrease to zero in a displacement equal to the track width (see section 3.3.2.3). From figure 4.25 the track width is calculated to be 0.598*mm*. This gives a value of -0.56 for ß, for this set of tests (i.e. these two interfering signals).

Beta is a complex function, and is not simply the correlation between the two signals. The correlation between the two signals does not take into account the signal processing applied to the signals to produce an error rate. Recall (from section 3.3.2.1) that there are worst-case combinations of transitions. A feature in the

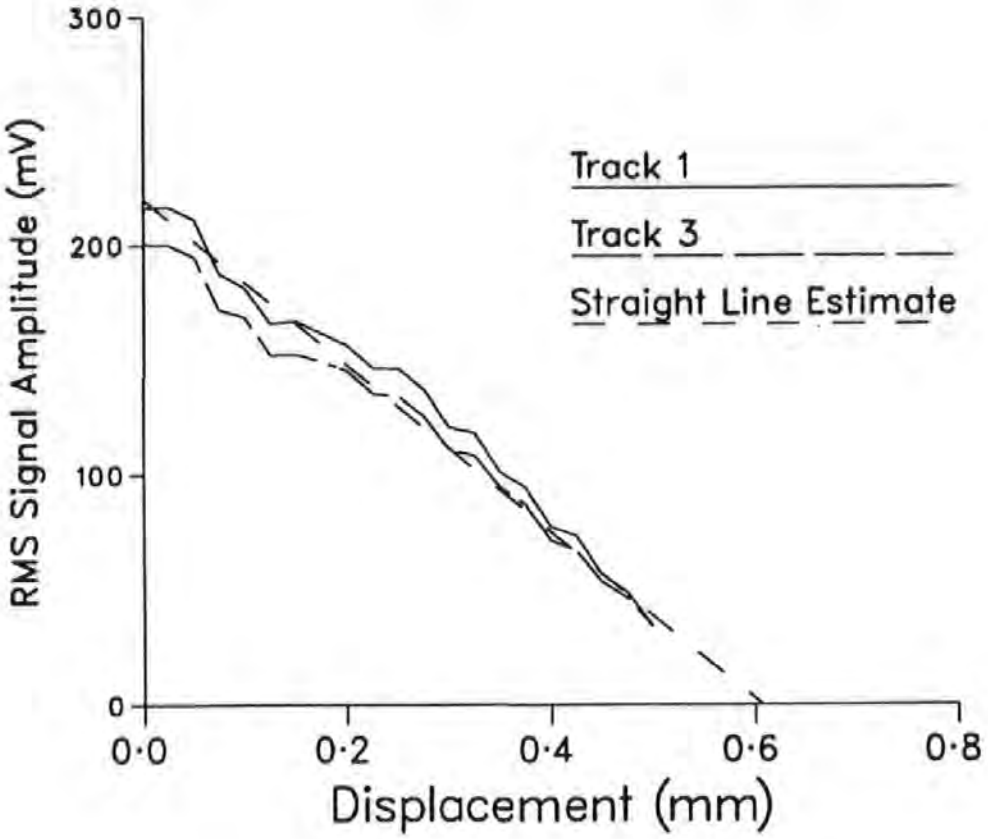


Fig. 4.25. The Effect of Lateral Head Displacement on Signal Amplitude.

signal of an adjacent track may have a considerable impact on the error rate if it interferes in such a worst-case region. To illustrate this, figure 4.26 plots values of m_{c1} and m_{c2} against data skew between two interfering signals.

When there is less than 0.3 data bits of skew between tracks, the error rate is always 'acceptable' (hence, the curves do not start at zero skew). As expected both curves show maxima at skews corresponding to integer multiples of the bit spacing i.e. when transitions for one track align with those of the interfering track.

The model was used to produce the simulated LHD error profile. Figure 4.27 shows the results from this simulation (compare with figures 4.15 and 4.16). As can be seen the overall error rate profiles of the two graphs are similar. There is however significant difference between the values for the first critical displacement,

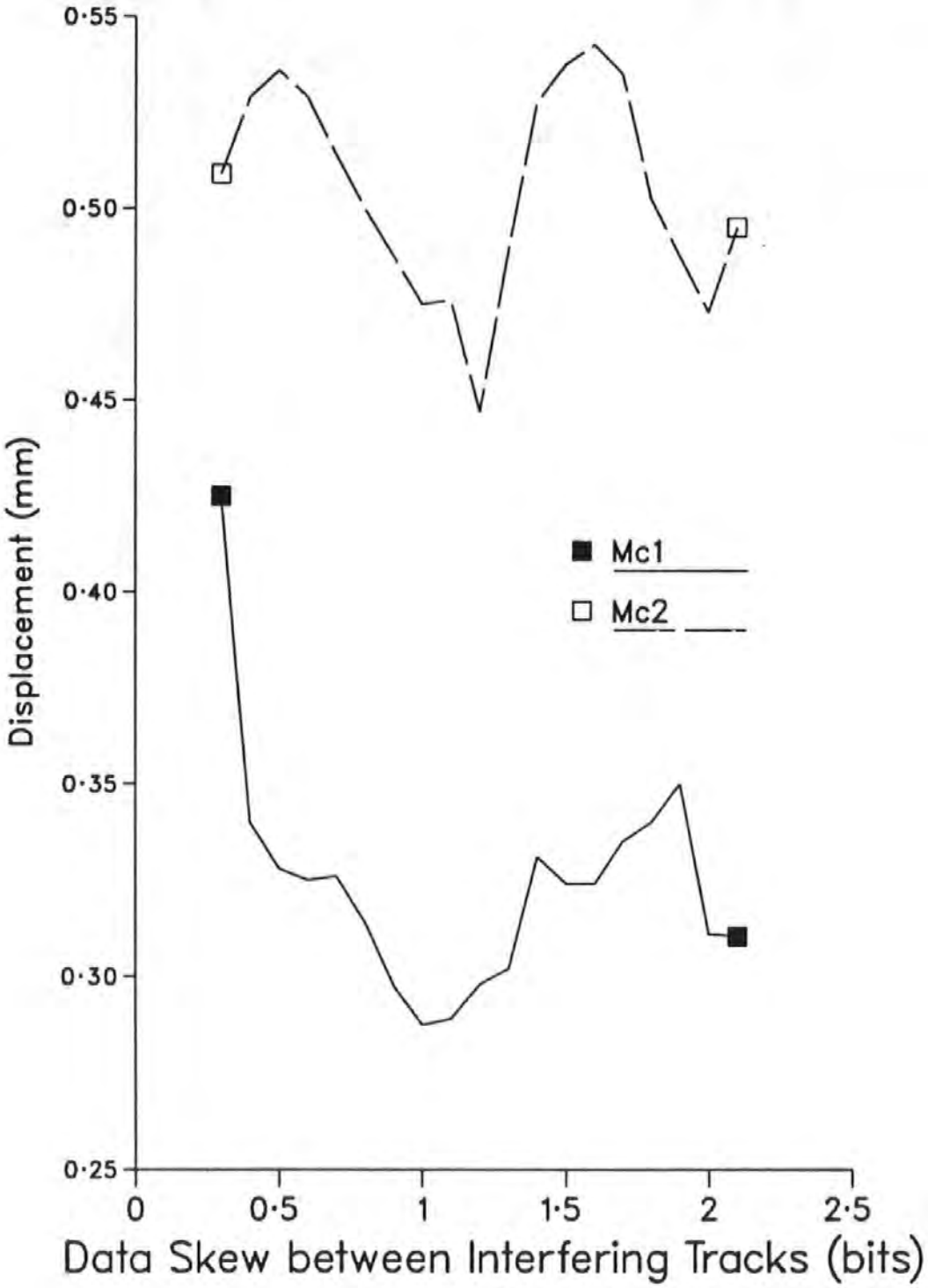


Fig. 4.26. The Simulated Effect of Data Skew between Tracks on the Maximum Displacement for an Acceptable Error Rate.

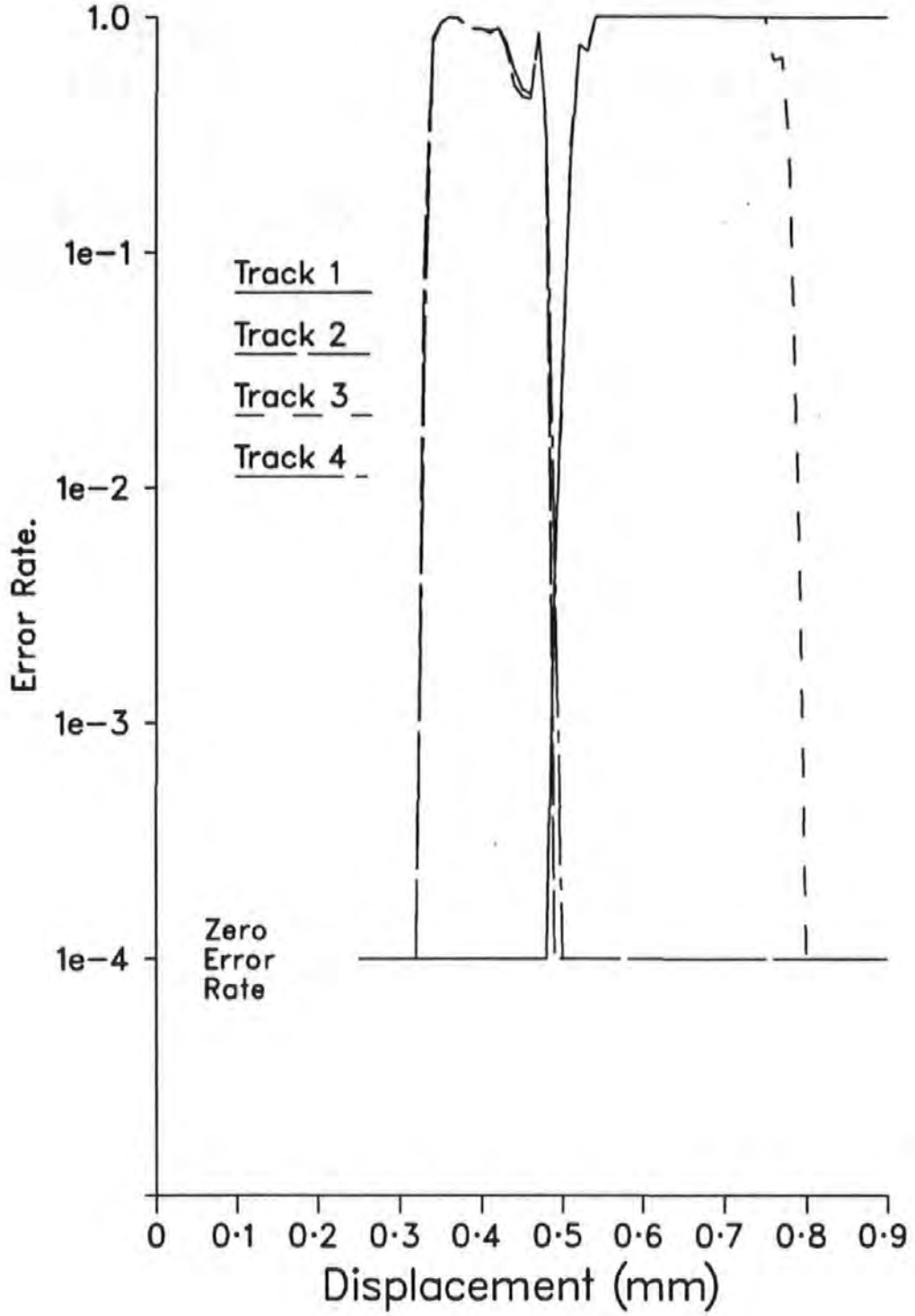


Fig. 4.27. The Simulated Effect of LHD on Error Rate.

m_{c1}, see Table 4.3.

	Compact-Cassette System	Model	Error
m_{c1}	0.400	0.316	-21%
m_{c2}	0.483	0.510	+5.6%
m_{c3}	0.457	0.474	+3.7%
m_{c4}	0.847	0.806	-4.8%

Table 4.3. Comparison of Critical Displacements between the Compact-Cassette System and the Model. (Dimensions in mm)

No suitable explanation could be found for this discrepancy. However, using Ferro-fluid and an optical microscope, faint signs of recorded information in the guard-bands could be observed on some cassettes (not necessarily those used for the above set of tests). This may be due to incomplete erasure, or some other mechanism and would obviously affect any displacement experiments.

The range of displacements from which an acceptable level of performance may be achieved was 76.5% and 74.1% (for the two track separations). These figures compare with 91.2% and 70% for the compact-cassette system. Whilst 74.1% compares well with 70%, the significant difference between 76.5% and 91.2% highlights the difference in the values of m_{c1} for the two system. Further investigations would be required to identify the source of this discrepancy.

The basic isolated pulse was changed to that for the MR head, and the model was used to simulate the displacement error profile for the MR head. Figure 4.28 shows the results of this simulation. A data rate of 515bps was used as this corresponds to the same rate as the compact-cassette system when normalised to the PW50. As the track separations are consistent, only one set of curves are produced. Extrapolating the curves to predict the critical displacements at an error rate of $1x10^{-5}$, produces figures of,

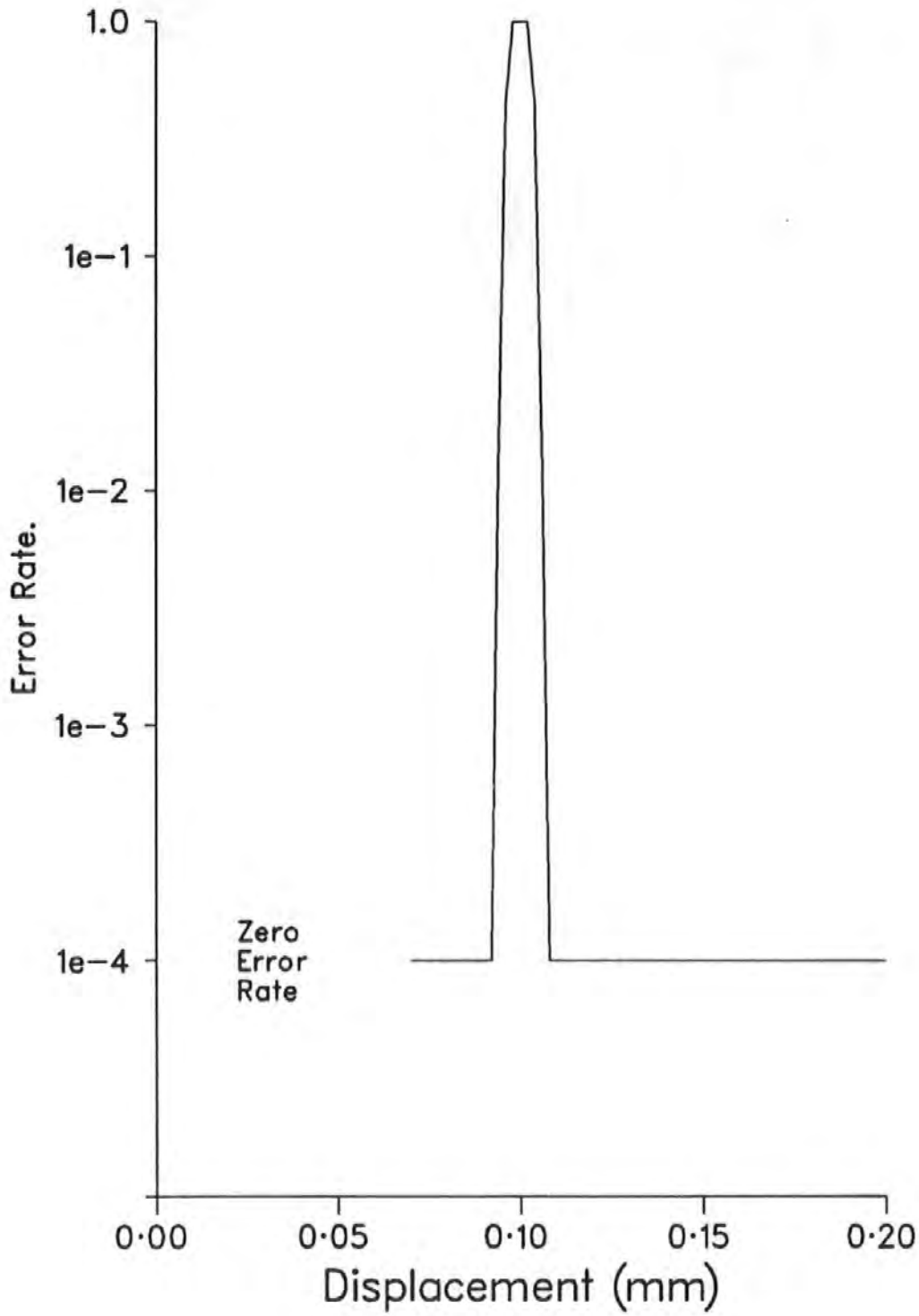


Fig. 4.28. The Simulated Effect of LHD on Error Rates for the MR Head.

$$m_{c1} = 0.091$$
$$m_{c2} = 0.109$$

From these figures, the range of displacements over which an acceptable level of performance may be achieved is 90.8% This compares with approximately 75% for the simulated compact-cassette system, and demonstrates the performance benefits of a write-wide, read-narrow design in the presence of LHD.

4.4.6. Lateral Head Displacement and Azimuth Skew.

From results previously presented, up to 20 minutes of arc of azimuth skew or 0.4mm of LHD may be introduced without a significant increase in error rate. However under normal operating circumstances both of these error causing sources may be present. Figures 4.29 and 4.30 illustrate the effect of LHD on error rates in the presence of 13.3 minutes of arc of azimuth skew. Thirteen minutes of arc was significant enough for its effect to be observed when combined with LHD, without directly causing errors.

In comparison with figures 4.15 and 4.16 (with zero azimuth skew), the curves of figure 4.29 and 4.30 fluctuate considerably more. Table 4.4 compares the 4 critical displacements with those for zero azimuth skew.

	Zero Azimuth Skew	Azimuth Skew of 20 mins of arc
m_{c1}	0.400	0.323
m_{c2}	0.483	0.656
m_{c3}	0.457	0.360
m_{c4}	0.847	-

Table 4.4. Critical Displacements, with and without Azimuth Skew.
(Dimensions in mm)

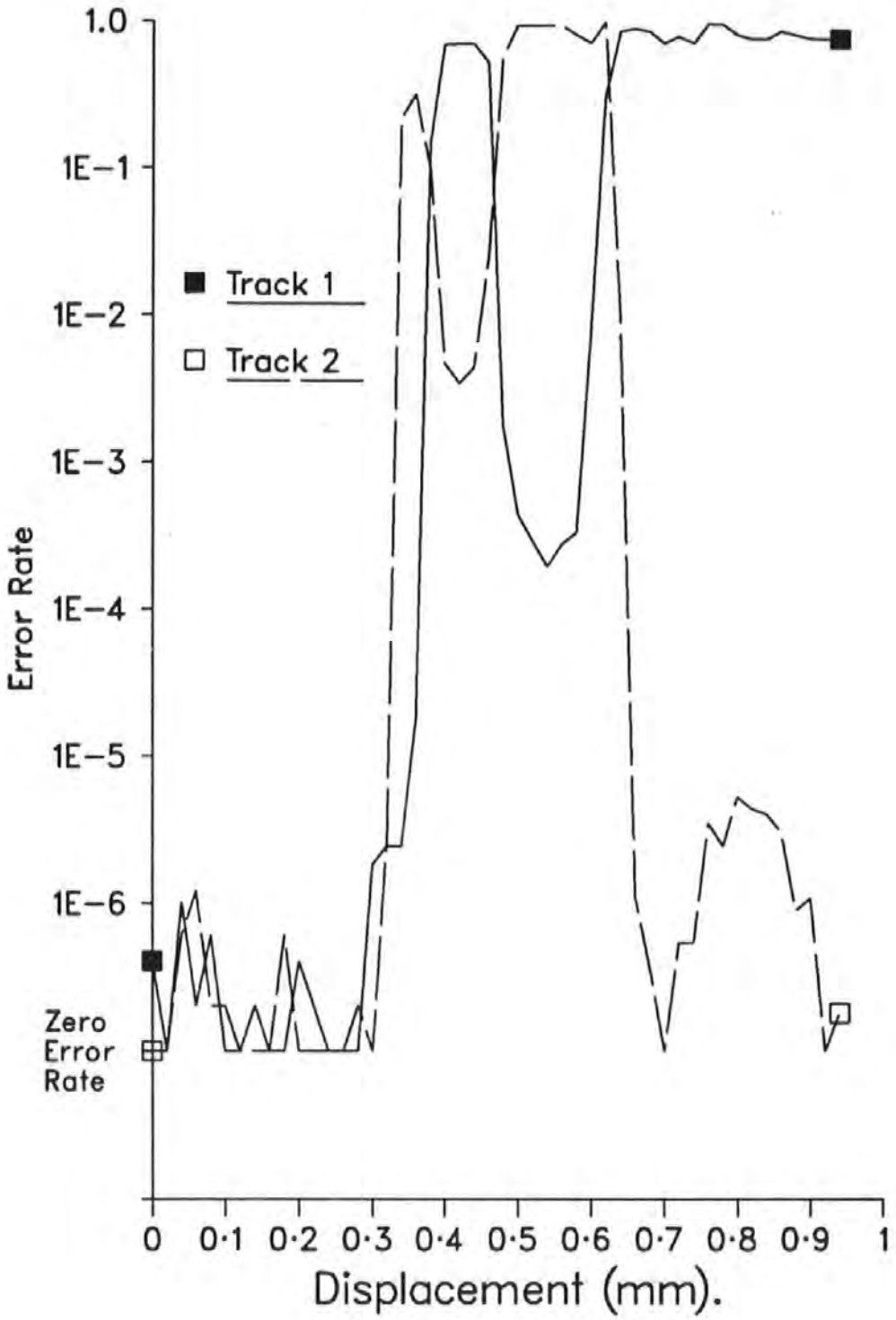


Fig. 4.29. The Combined Effect of LHD and Azimuth Skew on Error Rates (Tracks 1 & 2).

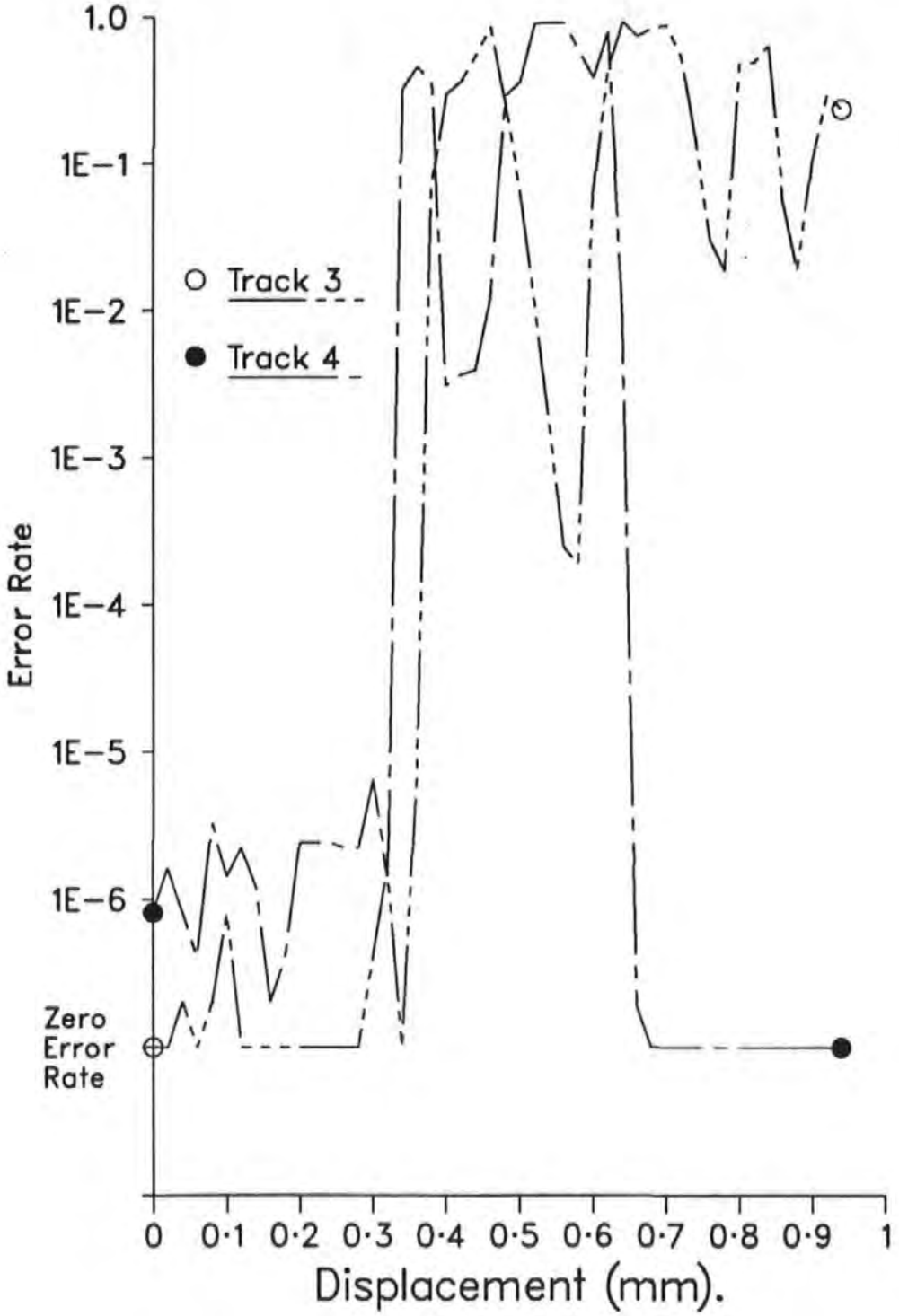


Fig. 4.30. The Combined Effect of LHD and Azimuth Skew on Error Rates (Tracks 3 & 4).

As expected, the performance degradation caused by the azimuth skew results in the onset of errors (at m_{c1} and m_{c3}) at smaller displacements than for zero azimuth skew. The range of displacements from which an acceptable performance may be achieved has been reduced from 91.2% to 66%. This degradation in performance is also assumed to be responsible for head 3 being unable to read track 2 at the required level of performance for the calculation of m_{c4}. The distinct difference between m_{c1} and m_{c3} again shows the effect of interference from an adjacent track.

However, both m_{c1} and m_{c3} are approximately equal to or greater than the inter-track guard-band width, and m_{c2} is greater than the track width. This suggests no interference occurred between tracks: the onset of errors at the critical displacements simply being due to the attenuation of the (degraded by azimuth skew) signal, caused solely by the displacement. No explanation can be found for this anomaly.

Also, at displacements of 0.56mm for head 1 and head 3 and at 0.42mm for head 2 and head 4 the error rate drops significantly. This suggests a non-linear mechanism in the interference process caused by LHD. No mechanism was found to explain this. However the excellent agreement between the graphs for track 1 and track 3, and the graphs for track 2 and track 4 strongly suggests the results were not spurious.

4.5. LHD Compensation Scheme.

This section presents the results for the LHD compensation scheme detailed in section 3.4. This scheme reconstructs the 'on-track' or optimal signal from the attenuated and corrupted signals of the displaced head. This requires a knowledge of the dimensions of the track format and the magnitude and direction of the LHD. The scheme uses the signal from the n^{th} head to calculate the n^{th} track component of the $(n-1)^{th}$ head signal. Due to inconsistencies between channels (e.g. in head efficiency, amplifier gain e.t.c.), the reconstructed signal will only be an estimate of the optimal

signal.

To determine the accuracy of the estimate, both the estimate and the actual signals ideally need to be captured simultaneously. This cannot be done as it requires the n^{th} head to be optimally reading the n^{th} track, whilst simultaneously spanning (and reading) the n^{th} and $(n-1)^{th}$ tracks: this is obviously physically impossible. Therefore, the optimal signal and the displaced signals from the compact-cassette system were captured on successive replays, and stored on the IBM PC's hard disc.

Figure 4.31 plots the maximum value of the cross correlation coefficient between the captured optimal signal and the uncompensated and compensated displaced signals. The cross-correlation coefficients were derived using a commercial DSP package (ILS, 1987) to calculate the Cross-Covariances, which were then normalised. Up to a displacement of 0.25mm the correlation coefficients for both compensated and uncompensated signals had the same value and remained very similar up to 0.35mm. This was as expected as adjacent tracks will not interfere until approximately 0.325mm. The SNR will be reduced by 3dB at approximately 0.3mm (w/2), and this may account for the slight divergence. Past this point the uncompensated signal correlation coefficient falls rapidly due to the increased interference from the adjacent track. The compensated signal's correlation coefficient does not decrease, remaining greater than 0.9. The underlying principle of the compensation scheme - that of removing the effect of an interfering signal - was therefore judged to be valid.

Figure 4.32 shows the effect of the compensation scheme on error rates when applied to the same captured waveforms. (Comparing figures 4.32 and 4.26 reinforces the fact that simple correlation measurements cannot be used directly to determine system performance). Figure 4.32 demonstrates the potential of the scheme. When the magnitude and direction of the displacement was known, the effect of LHD on error rate was completely compensated for. It is important to note that figure 4.32 includes the worst case condition, where the n^{th} head is half way between the n^{th} and $(n-1)^{th}$ tracks. Figure 4.33 shows the pre and post compensation scheme

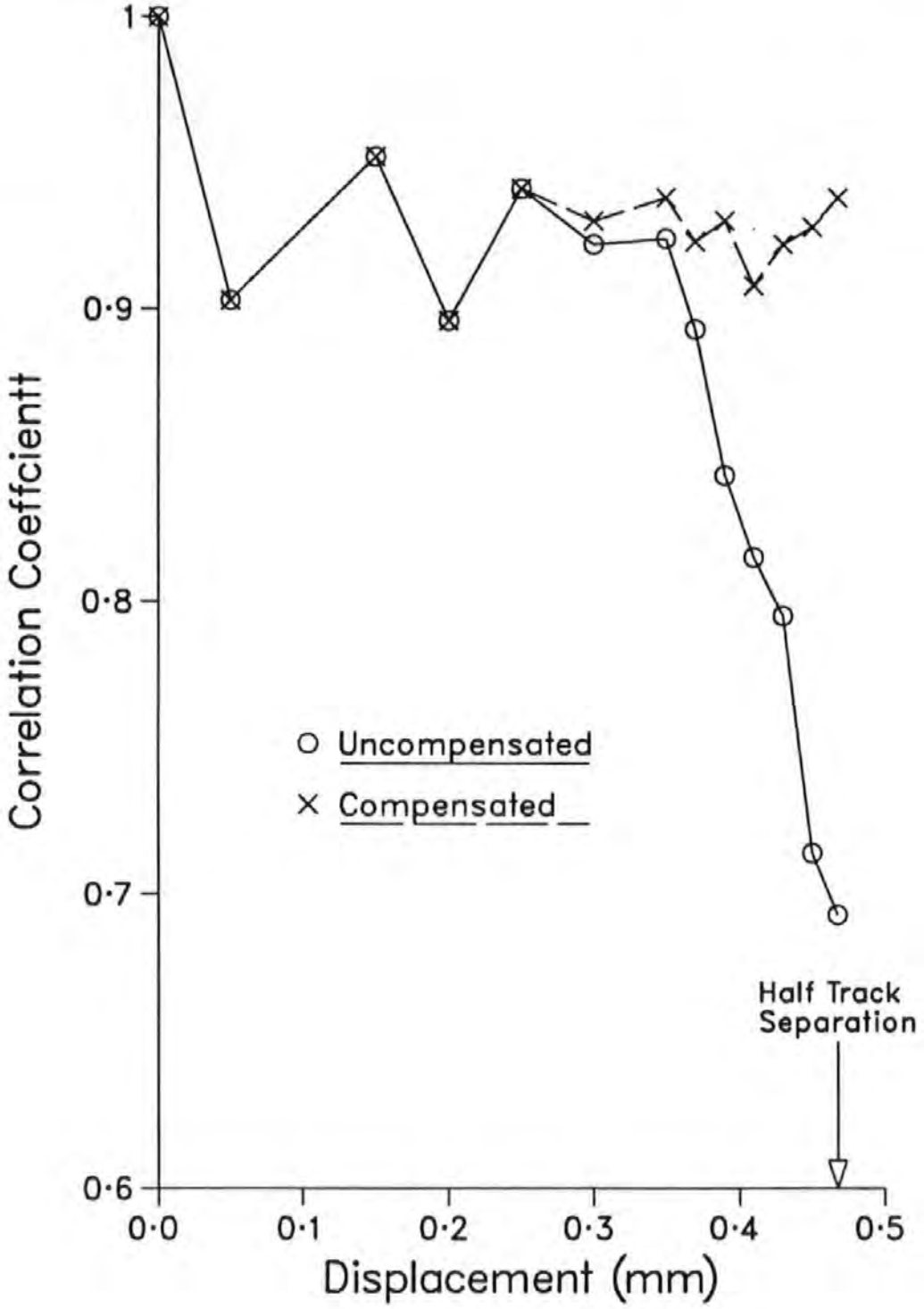


Fig. 4.31. The Efficacy of the Compensation Scheme on Interfering Track Signals.

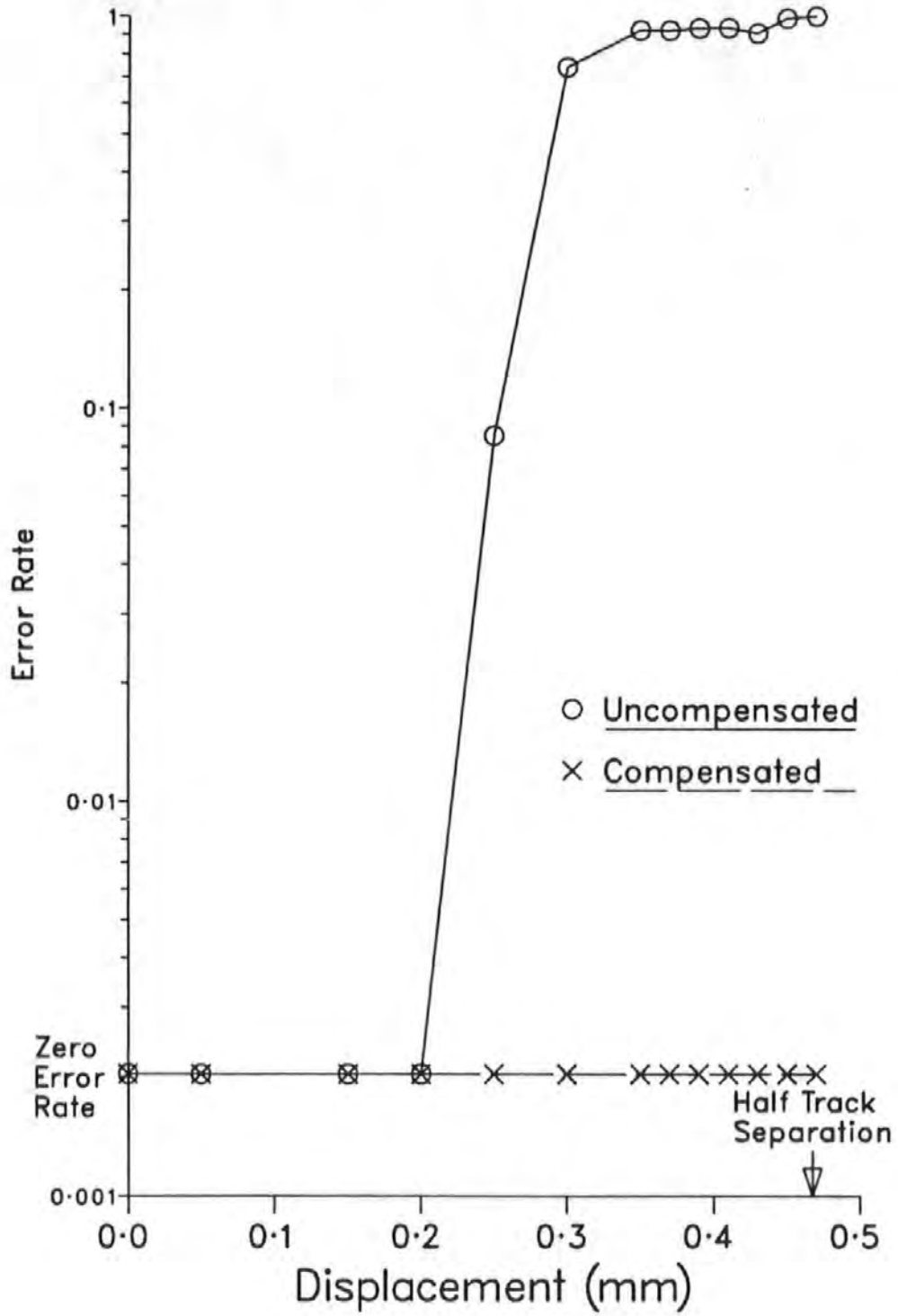


Fig. 4.32. The Effect of the Compensation Scheme on Error Rates.

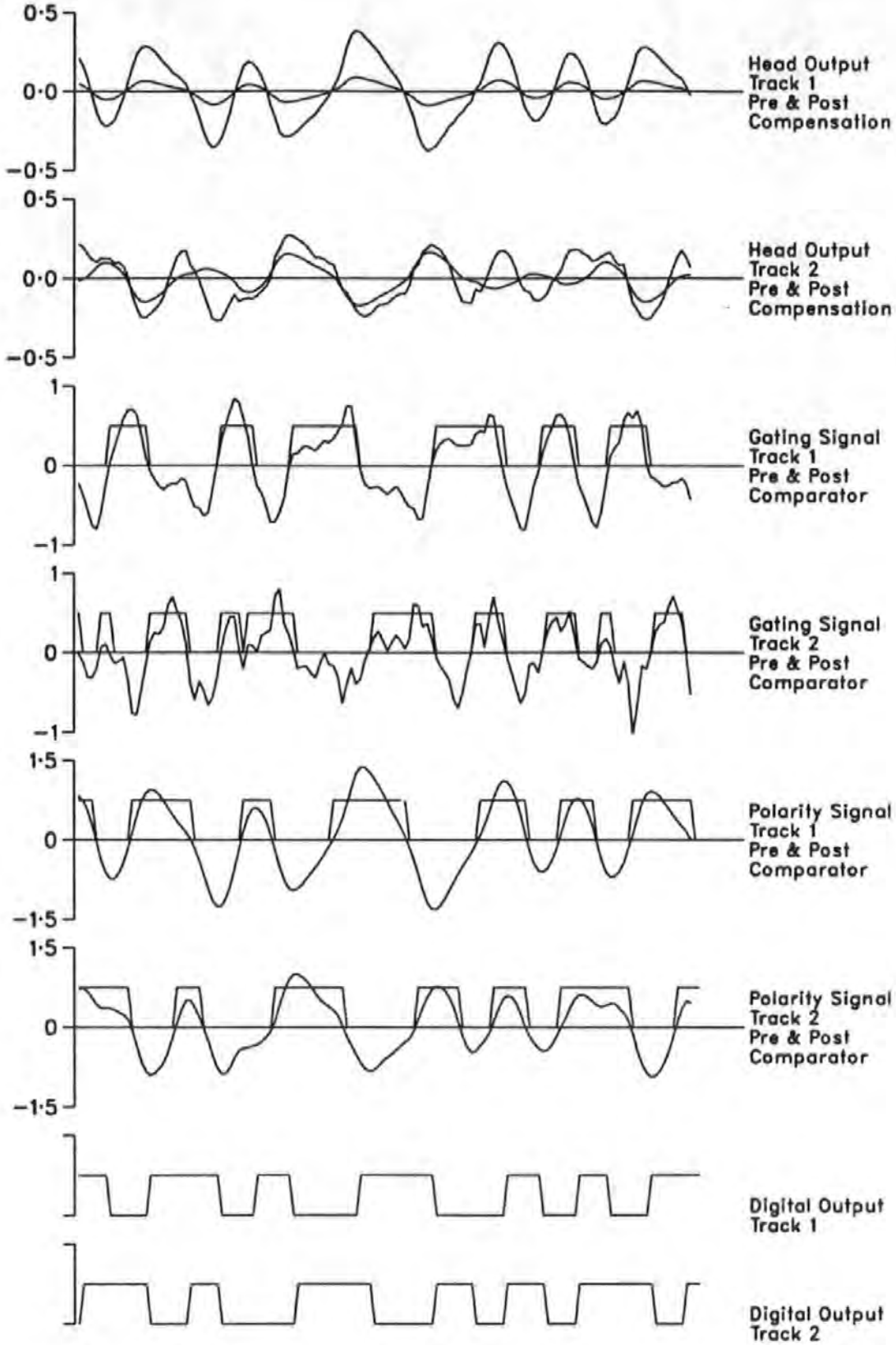


Fig. 4.33. Waveforms from the Model Compensating for LHD.

waveforms, together with the relevant GXO waveforms, whilst compensating for this worst-case LHD condition. Whilst the post-compensation signals show a large amount of noise, the GXO detector is able to successfully decode them.

Unfortunately, the exact magnitude of the displacement would not normally be known. Figure 4.34 shows how the compensation scheme performs when only an estimate of the displacement is known. For the 'RMS' curve, the displacements were calculated from the attenuation in the RMS value of the uncorrupted edge track signal, using:

$$\text{displacement} = \frac{\text{RMS - Offset}}{\text{Slope}} \qquad \text{Equ. 4.7}$$

where Slope and Offset are taken from the straight line estimate in figure 4.25. The two curves 'Optimal +10%' and 'Optimal -10%' show the error rate performance when the displacement used in the compensation scheme is +10% and -10% of the measured (assumed exact) value.

Figure 4.34 illustrates that the attenuation in the RMS of the signal is a good estimate of the displacement. It performed better than $\pm 10\%$ of the exact value. It is sufficiently accurate for the compensation scheme to considerably enhance the performance of the system. Assuming an error rate of 1×10^{-3} produces an acceptable level of performance, the displacement figure of merit was increased from 43% to 100% when the exact displacement was known, and from 43% to 92% when the level of signal attenuation was used to estimate the displacement. Also note, system performance was never degraded.

4.6. Limitation of the Amplitude Fluctuation Mechanism.

The mechanism used to introduce amplitude fluctuations (described in section 3.3.2.3) was found to be flawed. The amplitude fluctuation process is applied to the isolated pulses before being combined using LPS. The resultant waveform has a different amplitude distribution to that applied, see Table 4.5. Column one contains the data rate of the

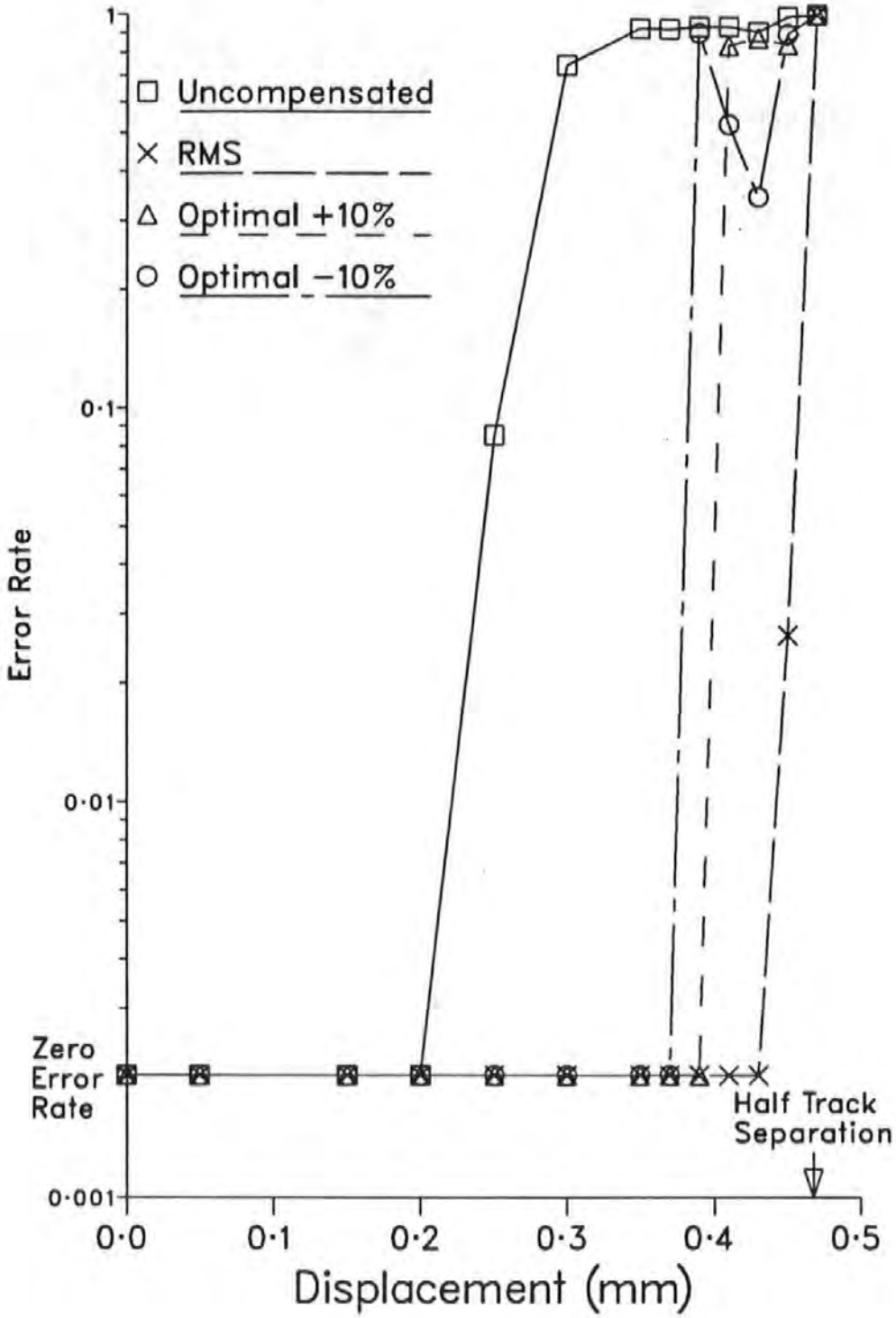


Fig. 4.34. The Effect of Estimating the Displacement on Compensation Scheme Performance.

(1) Data Rate.	(2) Basic Waveform.	(3) Actual Resultant.	(4) Desired. Resultant.	(5) Equivalent. Applied.
1000	0.0152	0.0332	0.0327	0.0295
2000	0.0997	0.1053	0.1039	0.0338
3000	0.1552	0.1606	0.1579	0.0413
4000	0.1933	0.1994	0.1955	0.0489
5000	0.2532	0.2610	0.2548	0.0633
6000	0.3350	0.3413	0.3363	0.0653

Table 4.5. Standard Deviations of Simulated Waveforms.

waveform. The second column contains the Standard Deviations (SDs) of the basic waveform, produced by the superposition of uniform height isolated pulses (i.e. zero deviation in amplitudes). The third column contains the SDs of the waveforms produced by the model, when the isolated pulse amplitudes had an SD of 0.029 (i.e. the value measured from the compact-cassette system, section 3.3.2.3). The fourth column contains the statistical addition of the SD of the basic waveform from column 2, plus an SD of 0.029. The fifth column is the statistically calculated value of SD that would need to be added to the basic waveform SD to produce the values in column 3. This fifth column would contain only the value 0.029 for a correctly implemented amplitude fluctuation mechanism, and therefore the figures in the column indicate the error in the implementation.

Table 4.5 also shows that as the data rate increases the distribution of amplitude fluctuations does not remain constant. The amplitude fluctuation process therefore needs to be modified so that it is applied to the complete waveform, and not to individual isolated pulses. Doing so would also allow drop-outs to be modelled more accurately in terms of duration and severity (e.g. Baker, 1977).

4.7. *occam* and the *transputer*.

All of the results presented in this chapter were obtained (to a greater or lesser extent) using the software described in chapter 3. This alone is felt to vindicate the choice of the ***occam*** programming language (and therefore the ***transputer***) for this project. This section highlights several key points and issues concerning the semantics of ***occam***, computational performance, and hardware and software architectures.

The semantics of ***occam*** greatly simplified many of the programming tasks. For example, once the Bi-Phase-L Channel Decoding process had been written, the code to run a number of them concurrently was simply,

```
PAR track = 0 FOR number.of.tracks
  ... Channel Decoder Process
```

The following code extends the above and could have been used to download (or PLACE) a number of Decoder processes onto a network of ***transputers***, where they would run in parallel.

```
PLACED PAR track = 0 FOR number.of.tracks
  PROCESSOR track T4
    ... Channel Decoder Process
```

The significance of the lines of ***occam*** above cannot be overstated. The first code fragment could be used to run 1 or 1000 or any integer number (known at compile time) of copies of the Decoder process concurrently on a single ***transputer***. The second code fragment could be used to distribute any number of Decoder processes over an array of ***transputers*** (each of type T4). The key point is: the ***occam*** for the Decoder process remains unchanged.

As a ***transputer*** array large enough to distribute the code over was not available, this facility could not be investigated. However, the code was written to take advantage of this facility had it become available. During simulation of the compact-cassette system, 35 processes run concurrently. This code could be transferred to a network of 35 ***transputers*** without a single modification to

the process's code. A '35 times' speed-up would not occur however as the processes are not balanced in terms of processing time i.e. the data rate through the network would be determined by the process with the longest processing time. Also, some simple message routing processes would need to be written as ***transputers*** have a maximum of 4 Links, and some of the processes need 8 channels. The second generation of ***transputers*** (the T9000 family (INMOS, 1991)) use automatic routing switches to avoid the need to use such extra processes.

The semantics of the language facilitates (or even encourages) a natural one-to-one mapping between the components of the compact-cassette system and those of the model. Although ***occam*** processes are not true 'objects' in the sense of an object oriented language (i.e. there is no inheritance mechanism or implicit encapsulation, once written they may be manipulated in an almost tangible manner.

The benefits this brings were most clearly demonstrated by the modification made to the GXO detector to perform the function of a Peak detector (section 4.4.2). The analogy between the modifications made to the ***occam*** code and the rewiring of the circuit was clear. The value or worth of the model is therefore increased as each process represents a basic building block that may be re-used elsewhere.

From this one-to-one mapping the programme is naturally formed into a Data Driven, Data Flow style of architecture (Williams, 1990). The code for the model formed two such data flow structures, one for the data generation and encoding (see figure 3.3) and a second for the data decoding and analysis (see figure 3.5). The data flowed through the processes where it was transformed. The analogy between the electrical signals flowing through the electronic circuitry is clear.

The performance of the data acquisition process was analysed in detail in section 3.2.2.1. A similar analysis was performed on the model's code. As a single 5kbps data bit was represented by 100 floating point values in the sampled data section of the model, the investigation concentrated on the nine concurrent processes that

constituted this section. Firstly, the overhead in terms of CPU execution cycles associated with the Data Flow structure was calculated. The format of, and channel protocol between these processes was standardised, and is shown in figure 4.35 along with the execution times.

```
PROTOCOL INT.OR.FLOAT
  CASE
    int ; INT
    float ; REAL32
:

SEQ
  WHILE running                                    9
    data.in ? CASE                                32
      int ; char
        IF
          char = terminate
            running := FALSE
          TRUE -- else,
            SKIP -- do nothing
      real ; data.to.be.transformed                26
        ... process data
        data.out ! real ; transformed.data         63
  ... pass on terminate.symbol
                                                Total 130
```

Fig. 4.35. Standardised *occam* process format
(Figures indicate execution times in CPU cycles).

As no suitable floating point value was available to be used as a 'terminate' token, a Variant Channel Protocol (INMOS, 1988a) was used. This allowed an integer terminate token to be passed down the same channel as the floating point sampled data. When a floating point number was input from the previous process, the relevant transformation was applied, and the new data output to the next process using the same channel protocol. When the terminate token was received, the main loop stopped running and the terminate token was passed to the next process. This ensured each process terminated correctly.

From figure 4.35, each sample passing through each process consumed 130 CPU cycles. As it takes approximately 20 CPU cycles to schedule a process, and each sample results in the process being

scheduled once, the total overhead per process incurred from the use of this Data Flow architecture was 150 CPU cycles per sample.

Secondly, the disassembled *occam* was searched for time consuming operations. These were determined to be the GWN generator and the use of floating point arithmetic. Table 4.6 shows the cost incurred (in CPU cycles) by the use of such operations for each process.

Process.	Overhead	GWN	Floating Point Arithmetic
Read (Liner Superposition)	150		645
Displacement	150		630
Head Amplifier	150		1290
GXO Differentiator	150		1490
GXO Gain Block	150		1490
GXO Gating Signal Comparator	150	5957	230
GXO Polarity Signal "	150	5957	230
GXO Gated Output	150		
Data Acquisition	150		
Totals	1350	11914	6005

Total 19269 CPU cycles

Table 4.6. Execution Times for the Processes used in the Model (in CPU cycles).

During simulation of a 5kbps waveform, each data bit took approximately 0.2 seconds to be processed. As each data bit was represented by 100 samples, each sample took 2mS to be processed, and therefore consumed 30000 CPU cycles (for a 15MHz part). Table 4.6 therefore accounts for the majority of the CPU's activity. (The total of 19269 cycles does not take into account the degradation in performance caused by slow external memory (see section 3.2.2.1.)).

From Table 4.6 it can be seen that the use of GWN to model electronic noise consumed in excess of 60% of the CPU's resources.

Whilst a figure of nearly 6000 cycles to generate each noise sample seems high, most GWN algorithms use trigonometric functions that are also very time consuming (for example sin(x) consumes approximately 3400 CPU cycles).

The use of floating point arithmetic consumed approximately 30% of the CPU's resources. This may be reduced by using a T800 *transputer* as it has a dedicated floating point co-processor (INMOS, 1988b). This reduces addition and subtraction from approximately 230 cycles to 7, and multiplication from approximately 200 cycles to 11. As floating point arithmetic was also used by the GWN generator, considerable improvements would be expected.

To verify this the model was transferred to a 17.5MHz T800 *transputer*. The time to simulate each data bit was reduced from 0.2 seconds to 0.099 seconds. Compensating for the difference in clock rates, this represents a 88% performance improvement.

In some programmes the communication overhead creates a performance bottle-neck. Referring back to figure 4.35, each process used 95 CPU cycles per sample (i.e. 32+63) inputting and outputing data. Communication therefore consumed less than 5% of the CPU's resources, and suggests little would be gained by improving the efficiency of the message passing (for example by increasing the length of messages).

4.8. Summary.

This section summarises the results and discussions of the three systems investigated.

Analytic expressions were developed to describe the shape of the isolated pulses from the inductive and magneto-resistive heads. Although there was excellent agreement between the shapes of the pulses produced by these equations and those from the heads, the technique used to derive these equations had two limitations:

i) Due to the discontinuities in the waveforms, a large amount of human intervention was required to produce reliable results.

ii) With no theoretical basis, the equations derived could not

be manipulated in any manner that could be related to the recording process.

The model could therefore be usefully extended by adapting a more conventional approach to pulse shape modelling, for example one of the methods investigated by Loze (Loze, 1990).

The recording current level was adjusted to minimise the effects of intersymbol interference in the un-equalised channel of the compact-cassette system. A reduction in peak-droop by more than 50% was achieved, albeit at the expense of signal amplitude and therefore SNR. Further performance improvements were observed when more advanced (and more expensive) recording medium tapes (e.g. metal particle) were used.

The compact-cassette system achieved a total data rate of 22kbps at an error rate of $1x10^{-5}$. The simulated data rate results agreed closely with those from the compact-cassette system, the only notable deviation being a shift of approximately 300bps (or 6% of 5kbps) in data rate. The improved error rate performance of the model was attributed to the use of new tapes during characterisation of the reference pulse. Simulation indicated that the error performance of the compact-cassette system may be increased further by using a Peak detector. The data rate to error rate ratio maximum of $6.7x10^{10}$ compared favourably with a similar system (Donnelly, 1989). The performance of the MR head in terms of data rates was very disappointing. This was attributed to the lack of magnetic shields (the pole-pieces), and increased spacing loss caused by poor fabrication.

The correlation coefficient between the positional error profiles was found to be high (around 0.9) between replays, but low (around 0.1) between recordings. These findings discouraged an investigation into using a knowledge of a cassette's positional error profile from one recording to determine the error correction scheme applied during subsequent recordings.

The strategy of processing the data from each track separately proved to be successful in eliminating errors caused by misaligned data due to azimuth skew. The effect of azimuth skew in terms of its

signal attenuating property is dependant on the recorded packing density. At 5kbps, azimuth skew in excess of 20 minutes of arc needed to be introduced before the attenuation was sufficient to directly cause errors. This represents considerably more skew than the 4.4 minutes of arc encountered under normal operating conditions (Donnelly, 1989). The investigation concluded that azimuth skew was not a severe limitation on the ultimate performance of the recording channel.

In the compact-cassette format, the amount of Lateral Head Displacement (LHD) encountered under normal operating conditions does not significantly contribute to the error rate. Significant levels of LHD were deliberately introduced for investigative purposes. From the LHD error rate profile the critical displacements that mark the boundaries between acceptable and unacceptable performance were noted. From these figures, the track separations and mutual interference factor between interfering tracks were calculated. The mutual interference factor was found to be highly dependant on the relationship between the two interfering signals, and the relationship was found to be complex.

A new figure of merit was proposed. It specifies the range of displacements from which an acceptable level of performance may be achieved, expressed as a percentage of the track separation. The compact-cassette system's LHD figures of merit were 91.2% and 70% for track separations of 0.888mm and 1.304mm respectively. From simulation, the proposed magneto-resistive head achieved an LHD figure of merit of 90.7%.

The simulated LHD results were in general agreement with those from the compact-cassette system, but only to a first approximation. The difference was highlighted by the LHD figure of merit, for which the model produced figures of 76.5% and 74.1% (compared to 91.2% and 70%).

Although unaffected by the data misalignment caused by azimuth skew, the signal attenuating effects of azimuth skew degraded the performance of the compact-cassette system in the presence of LHD. The LHD figure of merit was reduced from 91.2% to 66% (the second figure of merit could not be calculated as head 3 never achieved an

acceptable level of performance reading track 2).

The LHD compensation scheme was found to be successful in terms of estimating the on-track signal from the off-track signal. For all displacements investigated, the correlation coefficient between the on-track signal and the compensated off-track signal was maintained at a value greater than 0.9, including the worst-case displacement of half the track separation. The compensation scheme was sufficiently effective to allow the off-track signal to be decoded with no errors (again, including the worst-case displacement). The RMS value of the signal was found to be a good estimate of the magnitude of the LHD, performing better than $\pm 10\%$ of the exact displacement, and improving the LHD figure of merit from 43% to 92%.

The choice of ***occam*** and the ***transputer*** was vindicated by the results presented in this chapter. Solutions to inherently parallel problems were implemented in a very straightforward way, and efficiently executed by the ***transputer***.

4.9. References for Chapter 4.

BAKER, W.R., A Dropout Model for a Digital Tape Recorder. IEEE Trans. on Magnetics, Vol. MAG-13, No. 5, September 1977.

DONNELLY, T. Real-Time Microprocessor Techniques for a Digital Multitrack Tape Recorder, Ph. D. Thesis, Polytechnic South West, 1989.

JEFFERS, F., & Karsh, H. Unshielded Magnetoresistive Heads in Very High-Density Recording. IEEE Trans. on Magnetics, Vol. MAG-20, No. 5, September 1984.

ILS, Interactive Laboratory System, Signal Technology, Inc., California, USA, 1987.

INMOS Ltd., Occam 2 Reference Manual. Prentice Hall International (UK) Ltd., 1988 (a).

INMOS Ltd., Transputer Reference Manual. Prentice Hall International (UK) Ltd., 1988 (b).

INMOS Ltd., The T9000 Transputer: Product Overview. INMOS Ltd., 1991.

KATZ, E.R., & Cambell, T.G. Effect of Bitshift Distribution on Error Rate in Magnetic Recording. IEEE Trans. on Magnetics, Vol. MAG-15, No. 3, May 1979.

LOZE, M.K., Middleton, B.K., Ryley, A, and Wright, C.D., A Comparison of Various Methods for Characterising the Head-Medium Interface in Digital Magnetic Recording. IEEE Trans. on Magnetics, Vol. MAG-26, No. 1, January 1990.

WILLIAMS, S.A. Programming Models for Parallel Syatems. John Wiley & Sons Ltd., England, 1990.

5. Review and Conclusions.

The impetus behind the information storage industry is to increase the storage capacity of devices. In magnetic recording this means increasing the areal bit packing density. This may be achieved by reducing the space between, and width of, recorded tracks, and/or reducing the wavelength of the recorded information. Section 1.1.1 detailed the advantages, in terms of SNR, of increasing the track density as opposed to reducing the wavelength of the recorded information. Techniques to deal with many of the problems exacerbated by the use of higher track densities have been developed.

The problems of low SNRs and high error rates may be alleviated by the use of more sophisticated coding schemes. The higher levels of static inter-track cross-talk may be compensated for (at least partially) electronically. The maintenance of correct registration between the tape and head has been achieved by manufacturing components of the tape and tape transport mechanism to tighter tolerances. This approach results in increased manufacturing costs. With the advances being made in microprocessor technology, the cost of computation is falling, and is expected to carry on doing so. It is therefore extremely relevant to investigate how software techniques may be used to compensate for these mechanical deficiencies.

This project has investigated the performance of a multiple-track digital magnetic tape system, concentrating on the problems that a tape transport mechanism manufactured to a low tolerance may be prone to. The following conclusions pertain to the performance of the compact-cassette system, the structure of the computer model developed and the impact of ***occam*** and the ***transputer*** on the investigation.

The Compact-Cassette System.

The successful integration of the ***transputer*** into the data channel of the compact-cassette system was largely due to the

operation of the interface between the replay electronics and the *transputer*'s Link. Not only did it provide a seamless join between the hardware and software, it reduced by a factor of 9.5 the number of read operations required, compared to using a software polling technique with the same timing resolution.

The error classification scheme devised provided significantly more detailed information about the error rate profile than the raw bit error count used by many researchers. Its operation is not dependent on any specific data sequence, and only requires the specification of two parameters - the minimum good sequence length between bits in error, and the maximum bad sequence length.

The investigation into the effects of mechanical deficiencies on performance concentrated on Azimuth Skew and Lateral Head Displacement (LHD), the problems of inconsistent tape speed having been addressed (Donnelly, 1989). Misalignment of data between tracks of more than 1/4 of a data bit causes errors in a parallel sampling tape recorder. By reversing the overall architecture from a sequentially processed, N-bit parallel data channel, to N serial data channels processed in parallel, the system was unaffected by such data misalignments. Levels of azimuth skew greatly in excess of those encountered during normal operating conditions needed to be introduced before the resultant attenuation in signal amplitude directly caused errors. For these two reasons, combined with the fact that attenuation due to azimuth skew reduces with track width, azimuth skew was not viewed as a severe limitation on performance.

The impact of LHD on error rates increases as track widths and guard-bands decrease, and was therefore of prime interest. The compact-cassette's track dimensions are significantly larger than the magnitude of LHD encountered under normal operating conditions. Artificially high levels of LHD were therefore introduced to simulate the effects of LHD on very narrow track systems.

A system designer's objective is to keep the error rate within the capabilities of the error correction scheme. There is therefore a maximum error rate, above which an acceptable level of performance may not be attained. As the head is displaced from its on-track position, the system's performance (in terms of error rate) decreases

until it becomes unacceptable. To date, the system designer has used this point of unacceptability to specify the maximum displacement that may be allowed. If this approach is maintained, then as track dimensions reduce, system performance will become even more dependent on the mechanical tolerances of the tape transport mechanism. The approach used for this investigation was to assume that significant levels of LHD will be present in future systems, and to develop schemes to cope with the resultant degradation in performance. If the head is displaced past the point of 'acceptability', it will eventually start to read the track recorded by an adjacent head. With this approach, the LHD error rate profile forms three distinct regions:

i) Acceptable performance reading the correct track.

ii) Unacceptable performance.

iii) Acceptable performance reading an adjacent track.

A new figure of merit was proposed, one that specifies the percentage of displacements from which an acceptable level of performance can be achieved. For the two track separations the compact-cassette system achieved figures of merit of 91% and 70%. If this figure could be increased to 100%, LHD would no longer be a constraint. Using conventional decoding techniques such a figure is not possible: when the head is equidistant between two tracks neither track will be decodable.

A compensation scheme was devised to improve this figure. The scheme is effectively the inverse of the that used to introduce LHD in the model. From a knowledge of the track format dimensions, and an estimate of the LHD, the relative proportions of the signals from the two interfering tracks may be calculated. Starting with the signal from the head not corrupted by an adjacent track, the levels of interference may be calculated, and (at least partially) compensated for. Using this scheme, the correlation coefficient between the optimally replayed signal and the compensated signal from the displaced head remained greater than 0.9 (including the worst-case displacement of half the track separation). In terms of error rates, the compensation scheme was found to be 100% effective

in compensating the two outermost tracks when the exact amount of LHD was known. The attenuation of signal amplitudes may be used to estimate the magnitude of the LHD. Using this estimate, the compensation scheme performed better than using $\pm 10\%$ of the exact value of LHD.

Although, as stated above, azimuth skew was not viewed as a severe limitation on performance, the introduction of 13.3 minutes of arc of azimuth skew had a considerable effect on the LHD error profile: the first LHD figure of merit was reduced from 91.2% to 66%, whilst the second was immeasurable. Although such high levels of azimuth skew would not normally be encountered under normal operating conditions, it is recognised that azimuth skew compounds the problem of LHD, and would place a greater demand on a LHD compensation scheme.

The investigation established that the effects of azimuth skew need not impose a significant limitation on the performance of multiple-track digital magnetic recording systems. However, LHD was found to impose a severe limitation on the performance of multiple-track system that employ narrow and closely spaced tracks. The effects of LHD were quantified, and a compensation scheme devised to reduce its effect on system performance. This allows low-cost tape transport mechanisms to be used with narrower and more closely spaced tracks, thereby increasing the packing density.

The Model.

The model allows investigations into the performance of a multiple-track digital magnetic tape system to be carried-out in a strictly controlled environment. Although based on one specific recording system (the compact-cassette system), its modular structure greatly simplifies the task of adapting it to simulate different systems.

The isolated pulses that form the basis of the linear superposition process where derived analytically: no knowledge of any magnetic recording parameter was required. The high level of accuracy

of the results facilitated by this technique was gained at the expense of flexibility: the pulses cannot be manipulated in any manner that can be related to the magnetic recording process.

A one-to-one mapping between the physical elements of the compact-cassette system and the model was used whenever possible. For example, rather than simulate one track of the multiple-track system and assume consistency between tracks, all tracks were simulated concurrently. Therefore when the model was extended to allow lateral head displacement to be simulated, the interfering signals from adjacent tracks were already available. All that was required was an ***occam*** process to introduce the relevant amount of cross-talk.

This one-to-one mapping also allowed the elements of the model to be manipulated in an almost tangible way. This was demonstrated by the way elements of the GXO detector were rearranged to form a Peak detector. A direct analogy was drawn between the editing operations involved in modifying the model's code, and the rewiring operations that would have been carried-out, had the modifications been made to the electronic circuit.

As the whole of the recording system was included in the model, the effect of a parameter on performance was measured in terms of its effect on the error rate profile of the system. The error rate is the primary measure of a systems performance. Measurements based on other parameters, such as frequency response or SNR, are essentially intermediate indicators of performance. As the model has been validated against a real recording system, the signal to error rate conversion was known to be reliable and accurate.

The ability to simulate the performance of a complete multiple-track digital magnetic tape system in such an explicit manner, and the ability to measure the performance directly in error rates makes the model innovative.

The model was developed to provide a controlled environment in which investigations into the performance of multiple-track digital magnetic recorders may be carried-out. Its development also became a vehicle for a greater understanding of the recording channel. The inspiration for the LHD compensation scheme came whilst working on the process that introduces LHD in the model; one being the inverse

of the other. An advantage of the model's structure is that it may be modified easily, allowing novel ideas that aim to improve the performance of magnetic recording systems to be investigated.

occam and the *transputer*.

Parallelism occurs at many levels in a multiple-track digital magnetic recording system. The semantics of the *occam* programming language greatly simplified the design of the software written during this project. As *occam* does not require the real-world to be transformed into a sequential representation, the structure of the model was designed to follow that of the hardware. The model therefore naturally formed a Data Flow structure, with a near one-to-one mapping between the functional blocks of the hardware and the software.

The parallel streams of data associated with the multiple-tracks was described directly, as were the parallel data streams internal to the GXO detector model. A natural hierarchy of processes within processes was developed. The overhead incurred by the use of such a structure was found to be low. The usefulness of occam would be considerably undermined if parallel constructs, such as this, incurred a heavy performance penalty. However, the *transputer* executes *occam* processes very efficiently avoiding this. The level of efficiency achieved allows *occam* to be used even in the performance-critical area of real-time processing.

A significant benefit of using *occam* was that it greatly simplified the design of the code for both the compact-cassette system and the model. This was efficiently executed by the *transputer*. Many of the advantages that were gained from the use of *Occam* and the *transputer* in this project are equally applicable to other inherently parallel systems - not just limited to magnetic recording. Their use was a major contributory factor to the success of the techniques developed.

5.1. Suggestions for Further Work.

The isolated pulses used in the model developed were derived using purely numerical techniques. The usefulness of the model can be extended by using a pulse fitting technique that is based on the parameters of the magnetic recording process. This would allow the effect of such parameters on system performance to be included in further investigations.

Because of the compact-cassette's track dimensions it was not possible to investigate the performance of the LHD compensation scheme when applied to more than two tracks. Due to the iterative nature of the scheme, its efficacy is expected to decrease as the number of tracks increase. This is because the signal from the n^{th} track is estimated initially, and then, based on this estimate, an estimate of the signal from the $(n+1)^{th}$ track is made. As each estimate is based on all previous estimates, successive estimates will decrease in accuracy. The effect of this on the performance of the scheme needs to be investigated. The model may be used to perform such an investigation.

The performance of the *transputer* used during this investigation was insufficient to allow the data from the compact-cassette system to be processed in real-time. The performance of the next generation of *transputers* has been dramatically improved. Comparing the T414-15(INMOS, 1988) used in this investigation with a T9000-50(INMOS, 1991), instruction throughput has been increased 25 fold, whilst the floating point performance has been improved approximately 350 fold. The improvement in performance would speed-up simulations, and allow even more sophisticated schemes to be implemented.

The envisaged data rates of HDTV recorders demands the use of a multiple (or parallel) track format. Further investigations will need to be made as current systems lack the necessary performance. Tools and techniques such as those developed in this project may be used in such studies.

References for Chapter 5.

INMOS Ltd., Transputer Reference Manual. Prentice Hall International (UK) Ltd., 1988 (a).

INMOS Ltd., The T9000 Transputer: Product Overview. INMOS Ltd., 1991.

Appendix A: Published Paper.

Presented at EUROMICRO '88. Supercomputers: Technology and Applications, Zurich, August 1988. Published in Microprocessing and Microprogramming 25, pp281-285, by North-Holland.

North-Holland
Microprocessing and Microprogramming 25 (1989) 281 – 286

A Real-Time Transputer-Based System for a Digital Recording Data Channel

T.J. Jackson, D.J. Mapps, E.C. Ifeachor and T. Donnelly

Department of Electrical and Electronic Engineering, Plymouth Polytechnic, Drake Circus, Plymouth, Devon, PL4 8AA, U.K.

ABSTRACT: A number of techniques found useful when using a Transputer, programmed in occam, as the data processing element in the data channel of a four-track compact-cassette digital magnetic tape recorder are described. A simple circuit has been designed which multiplexes the asynchronous data from the four tracks down a single Link. The Transputer's ability to handle parallel processes, and to accurately time events has been utilised to solve the problems caused by tape skew.

1. INTRODUCTION.

Through the use of higher linear bit densities and a greater number of tracks, the areal bit density of magnetic digital storage systems has increased. Wider bandwidths and new coding schemes [1] have enabled data transfer rates to be increased. Coding is normally used to maximise the use of available bandwidth and to enable errors to be corrected. Our long term aim is to achieve further improvements using microprocessors in the coding process. Figure 1 shows the overall system.

There are a number of issues which must be addressed, such as limited bandwidth and the corruption of data. The data channel has a restricted bandwidth, with zero response at zero frequency and a limited high frequency response. Therefore, channel coding is normally used to match the characteristics of the data stream to that of the recording channel. Primary causes of errors include tape/head separation and tape-skew variation. Error correction coding [2] introduces extra information into the data stream in such a way that the data lost or corrupted during these occasions can be detected and corrected.

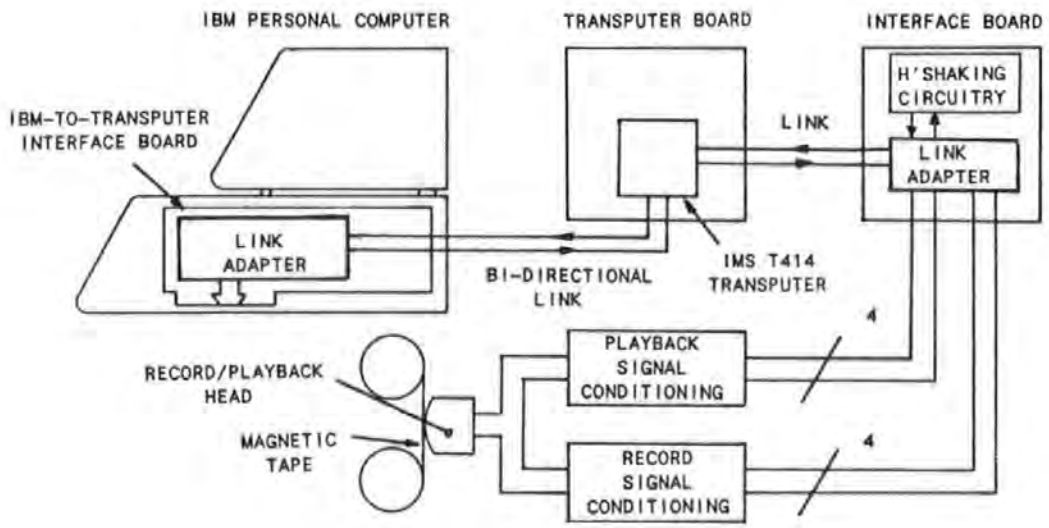


Figure 1: Transputer-Based Digital Recording System.

Many of the tasks involved in the recovery of data from multiple track digital recording systems are inherently parallel in nature. Therefore, when these tasks are to be performed by a conventional microprocessor, the description of these tasks must be written in a sequential form. There is often no efficient mapping from the natural parallel description to the sequential one. Consequently, the final implementation is often inefficient. This compounds the problem of limited processing time imposed by 'real-time' processing. As a result, systems often require dedicated hardware to carry out the coding. However, recent advances in VLSI fabrication techniques and architectures have raised the level of performance of some general purpose microprocessors, to the point where their use in the coding process may be realistically investigated.

Three techniques which have proved useful in our investigation are now described. Following a brief resume of the salient points of the Transputer is a description of a circuit which enables up to eight asynchronous data channels to be multiplexed down a single Transputer Link. Two software techniques follow: the first solves the problem of tape-skew at the decoding process, whilst the second removes some of the time dependent aspects of processing the data stream in real-time.

2. THE TRANSPUTER.

The Transputer's [3] high instruction throughput (10 MIPS for a 20 MHz IMS T414) and wide I/O bandwidth (40 Mbits per second) make it well suited to the evaluation of complex algorithms at high data rates. Of more importance to this application is the ease with which parallel systems can be described and efficiently evaluated in the Transputer's native language, occam. Transputers can support two levels of concurrency: true concurrency using more than one Transputer in a network, and pseudo-concurrency by time-slicing tasks within the Transputer. The main point is that the Transputer has a hardware scheduler. This is separate from the processing unit and controls the time-slicing of tasks. This has two benefits. The programmer does not have to write a task scheduler in software, and the processor is relieved of the extra processing involved in executing this code. As there is no syntactic difference between a parallel process being run truly concurrently or pseudo-concurrently, the occam programme can be written without specifying how many Transputers will be used to execute the final version (this is done at the Configuration stage).

3. INTERFACING.

The Transputer's primary method of communication is a two wire, bi-directional serial Link. Transputers have one or more Link interfaces which control the flow of data over the Link. Once initiated this Link transfer proceedes without further intervention from the processor. If communication is with another Transputer, the Link connects to that Transputer's Link interface. An adapter can be used to convert the Link to a more common format, enabling devices without Link interfaces to be communicated with. An IMS C011 Link Adapter [4] was used, which, in mode 1, converts the Link into two handshaken byte-wide parallel interfaces. To enable Links to comply with occam's channel communication protocol, the input interface has two handshaking lines: Input Valid (IVal) and Input Acknowledge (IAck). However, the data channel of a tape recorder is asynchronous and does not monitor or generate the required complementary signals.

Three solutions to this problem were considered. The signals may be generated by the corresponding handshaking lines of the output interface (QVal and QAck). This requires an output to be issued with the input command whenever data is to be read. This does not require extra hardware, but increases the processing time associated with inputting data, and may hinder the use of the output interface itself. Software polling must be used, and this is wasteful of the processor's resources, as well as violating occam's channel communication protocol. Secondly, a Port which does not use handshaking may be mapped into the memory map of the transputer. Ports are non-standardised, need extra circuitry and require software polling.

The third solution, the one chosen, was to build additional circuitry which monitors and generates the required handshaking signals. Figure 2 shows the circuit and timing diagram. When the magnitude comparator detects new data it takes IVal high. This disables the transparent latch stopping further new data appearing at the comparator. When the Link Adapter has read the data from the flip-flops it takes IAck high. This clocks the flip-flops, transferring the new data to the input of the interface for the next read, and making the P and Q inputs of the comparator equal. The comparator takes IVal low to complete the cycle. If new data had arrived whilst the latch was disabled, it would now initiate a new read cycle, else the circuit waits for new data. In this way up to 8 asynchronous data channels can be reliably multiplexed down a single Link.

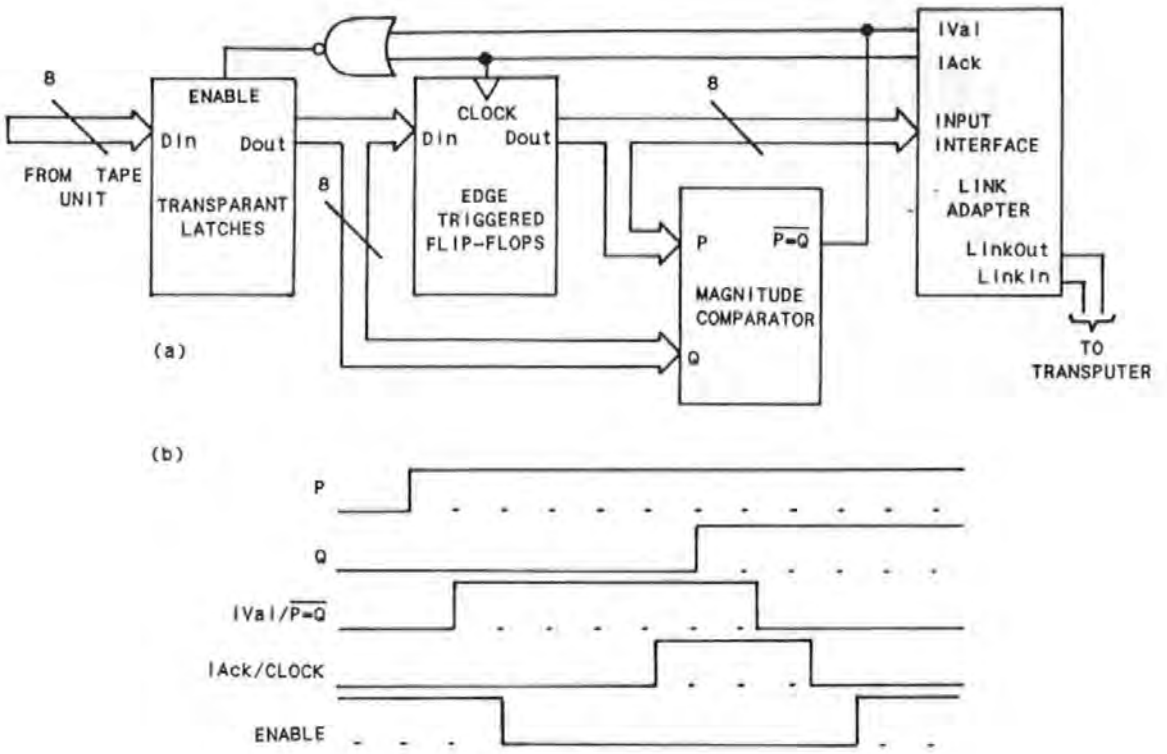


Figure 2: (a) Interface Circuit Diagram, and (b) Timing diagram.

Two points are worth noting. Firstly, the timing resolution between two non-coincident data transitions using this interface is dependent on the execution time of the process performing the input instruction. This process should be run at high priority, and (as with all high priority processes) kept as short as possible. Secondly, the flip-flops introduce a delay of one bit period between the time the data is detected by the comparator, and the time the data is read by the Link Adapter.

4. DE-SKEWING FOR CHANNEL DECODING.

The initial task of the channel decoding process is to detect transitions in the data stream, and from this decide at what point the data should be sampled. If all tracks are sampled at the same time, tape-skew needs to be taken into account. Ideally, the tape should pass the head in a straight line. In practice, due mainly to imperfectly slit edges, the tape weaves across the head, producing a constantly varying skew angle, see Figure 3.

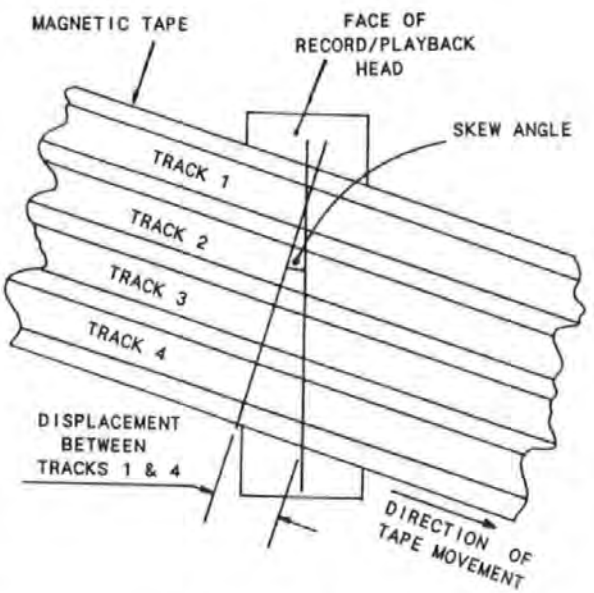


Figure 3: Tape Skew.

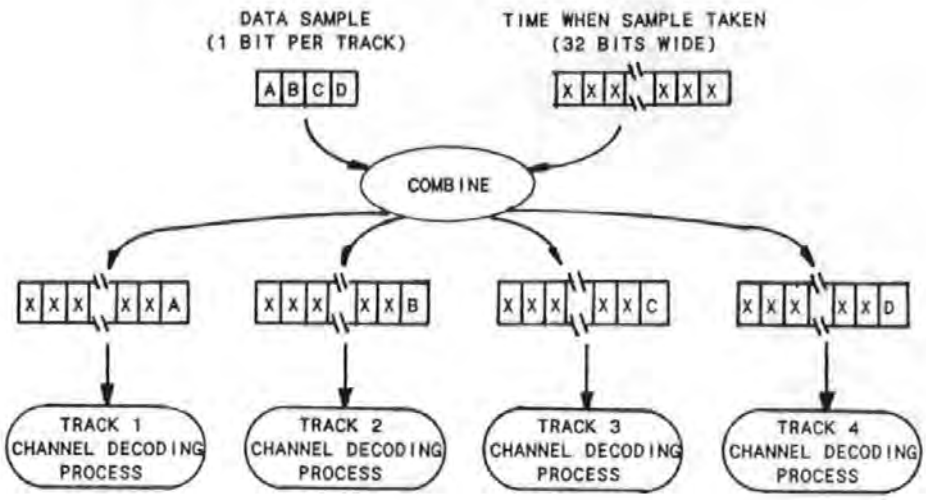


Figure 4: Time Stamping Data.

At low linear packing densities this is not a significant problem as the amount of skew is insufficient to cause the sample point to drift into the next bit cell. However, as the linear packing density increases, the magnitude of data-skew (measured in bits) between tracks increases. At high recording frequencies it has been shown [5] that tape-skew is a significant cause of errors.

This problem was addressed in software by having a separate channel deoding process for each track, thus exploiting the Transputer's parallel processing capability. The channel coding algorithm used was Biphase-Level. Each track has its clock information regenerated and sample points calculated independently of the other tracks. This results in a skew insensitive system which, as a result of the de-coding algorithm used [6], is self synchronising and insensitive to tape-speed variations.

5. TIME STAMPING DATA.

When a single Transputer is running a number of processes in parallel, although the logical result will be the same as though each process were running at the same time, only one process is being executed at any point in time. Therefore, if a number of parallel processes are timing external events, only one will correctly time the event.

As mentioned in the previous section, the channel decoding processes use the time interval between successive transitions in the calculation of the sample point time. As these processes (one for each track) are run in parallel, they cannot reliably record the time of the transitions. A solution is to have a seperate process, running at high priority, which samples all tracks simultaneously and records the time the sample was taken. The individual data bit for each of the tracks is then output to the relevant decoding process along with the sample time. Each decoding process can then calculate the time interval between successive transitions.

The smallest data type which can be output from a process is a byte. When the sampling process has a single data bit to output to a decoding process, it would be very inefficient if 8 bits had to be output. A more efficient solution is to combine the sample time data (a 32 bit integer) and the data bit, see Figure 4. By replacing the least significant bit of the sample time data with the data bit, both pieces of information can be efficiently sent to the decoding processes, albeit at the expense of a reduction in timing resolution.

Time stamping data in this way relieves some of the timing problems caused by processing in real-time, that is, the data will be correctly processed regardless of when this is carried out. This allows simple data buffers to be used to average out processing demands. It also helps in the programme development stage. It is much more difficult to debug a programme

running in real-time. After the data has been time stamped, it need no longer be processed in real-time. This allows the programme to be debugged in two parts, the majority of which can be done off-line, greatly simplifying the task.

6. CONCLUSIONS.

A simple method of multiplexing up to eight asynchronous data channels down a single Link has been devised. This, together with the de-skewing and time stamping techniques described, has enabled a Transputer to be successfully used as the processing element in the data channel of a compact-cassette digital magnetic tape recorder.

ACKNOWLEDGEMENTS.

The authors would like to thank the Science and Engineering Research Council, and Thorn EMI Central Research Laboratories for their financial support.

REFERENCES.

[1] Doi, T.T., Channel Codings for Digital Recordings, in: J. Audio Eng. Soc., Vol. 31, No. 4, pp.224-238. April 1983.
[2] Watkinson, J., The Art of Digital Audio. Chapter 7. (Focal Press, 1988).
[3] IMS T414 Transputer Data Sheet. INMOS Limited. August 1987.
[4] IMS C011 Link Adapter Data Sheet. INMOS Limited. August 1987.
[5] Donnelly, T., Mapps, D.J. and Wilson, R., Real-time Microprocessor Monitoring of Skew Angle in Compact Cassette Multitrack Magnetic Tape System, in: J. of I.E.R.E. Vol. 56, No. 2, pp 49-52, February 1986.
[6] Donnelly, T., Mapps, D.J. and Wilson, R., An Intelligent Microprocessor Interface for a Low-Cost Digital Magnetic Tape Recorder, in: Microprocessing and Microprogramming 23 (1988) pp333-338.

Appendix B: Mathematics of Magnetic Recording Theory.

This appendix is derived largely from the introduction given by Middleton (Middleton, 1987), with additions from Mallinsons book (Mallinson, 1987).

Figure B1 shows the pole-pieces of an inductive ring type head travelling over a magnetised section of tape. If the components of the magnetisation are M_x and M_y, and the fields from the head have components $H_x(x,y,z)$ and $H_y(x,y,z)$, then the components of the induced voltages can be shown to be:

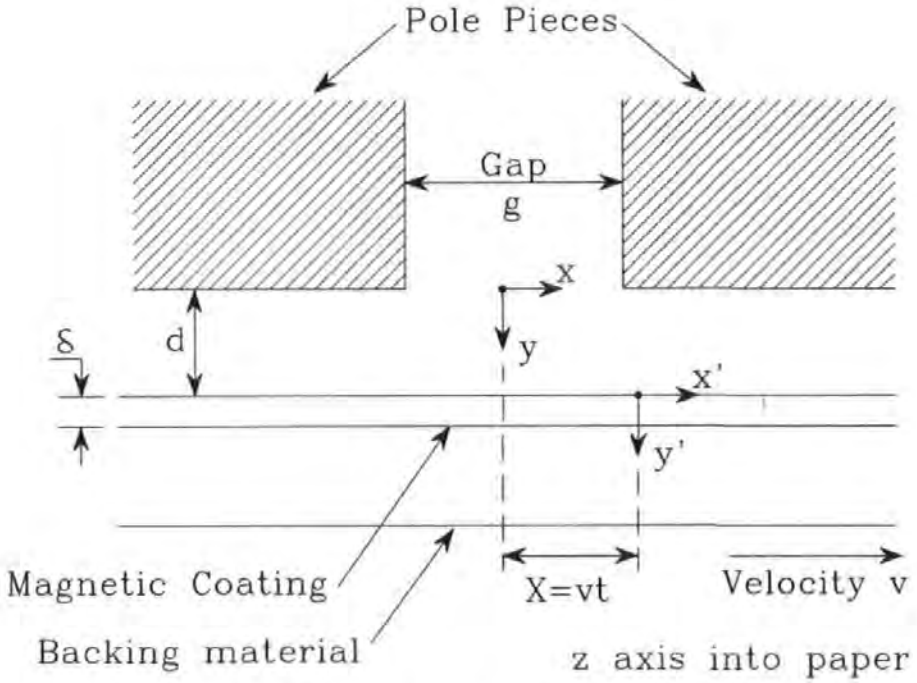


Fig. B.1. Coordinate System and Symbols of the Reciprocity Theorem.

$$e_x(X) = -\mu_0 V \int_{-w/2}^{+w/2} \int_{d}^{d+\delta} \int_{-\infty}^{+\infty} \frac{dM_x(x-X)}{dX} \frac{H_x(x,y,z)}{i} \, dx \, dy \, dz$$

Equ. B.1

$$e_y(X) = -\mu_0 V \int_{-w/2}^{+w/2} \int_{d}^{d+\delta} \int_{-\infty}^{+\infty} \frac{dM_y(x-X)}{dX} \frac{H_y(x,y,z)}{i} \, dx \, dy \, dz$$

Equ. B.2

where μ_0 = permeability of free space.

X = Vt

V = Tape-to-head Velocity

i = Current to produce above field

d = Head-to-Medium spacing

δ = Media Thickness

w = Track width (assumed equal to read track width)

In most cases the width of the recorded track is so large compared to other dimensions that the field in the z direction may be assumed constant. This simplifies the above to:

$$e_x(X) = -\mu_0 V w \int_{d}^{d+\delta} \int_{-\infty}^{+\infty} \frac{dM_x(x-X)}{dX} \frac{H_x(x,y)}{i} \, dx \, dy$$

Equ. B.3

$$e_y(X) = -\mu_0 V w \int_{d}^{d+\delta} \int_{-\infty}^{+\infty} \frac{dM_y(x-X)}{dX} \frac{H_y(x,y)}{i} \, dx \, dy$$

Equ. B.4

Expressions are now required for the magnetisation and head fields. From work by Karlqvist, for distances y > g/2, the head field components are:

$$H_x = \frac{H_g}{\pi} \left[\arctan\frac{(g/2)+x}{y} + \arctan\frac{(g/2)-x}{y} \right]$$

Equ. B.5

$$H_y = \frac{-H_g}{2\pi} \ln \left[\frac{((g/2)+x)^2 + y^2}{((g/2-x))^2 + y^2} \right]$$

Equ. B.6

where H_g is the field (assumed constant) in the Gap of width g.

If one assumes the magnetisation in the x direction varies sinusoidaly, i.e.

$$M_x(x) = M_0 \sin kx \qquad \text{Equ. B.7}$$

where M_0 = Maximum magnetisation
k = wave number = $2\pi/\lambda$
λ = wavelength of the recorded signal

then the reproduced signal is:

$$e(X) = C_1 Vw.\ k\delta.\ (e^{-kd}).\ \frac{(1-e^{-k\delta})}{k\delta}.\ \frac{\sin(kg/2)}{(kg/2)}.\ \cos(kX) \qquad \text{Equ. B.8}$$

Spacing Loss — Thickness Loss — Gap Loss

where $C_1 = -\mu_0 M_0 H_g g / i$

The three loss terms indicated are discussed in the main text.

In digital recording the Williams-Comstock model is often used, where (considering only the x components) the magnetisation between recorded bits is assumed to vary according to:

$$M_x = \frac{2}{\pi} M_0 \arctan \frac{x}{f_x} \qquad \text{Equ. B.9}$$

where f_x defines the length of the transition, and is refered to as the arctangent parameter, see figure B.2.

The reproduce signal now becomes:

$$e_x(X) = C_2 \int_0^{d+\delta} \left[\arctan \frac{((g/2)+X)}{(f_x+y)} + \arctan \frac{((g/2)-X)}{(f_x+y)} \right] dy \qquad \text{Equ. B.10}$$

where $C_2 = -2\mu_0 Vw M_0 H_g / \pi i$

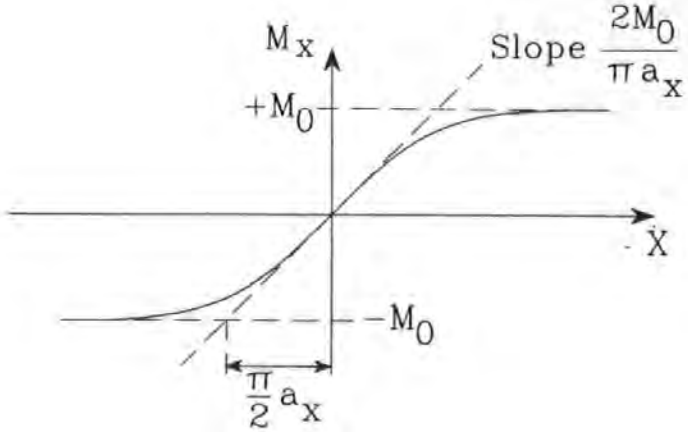


Fig. B.2. Arctangent Magnetisation.

This may be simplified further by assuming a 'near zero' gap head, thus:

$$e_x(X) = C_3.\ln\left[\frac{(f_x+d+\delta)^2+X^2}{(f_x+d)^2+X^2}\right] \qquad \text{Equ. B.11}$$

where $c_3 = \frac{-\mu_0 V w M_0 H_g g}{\pi i}$

Solving equation B.11 to find the Pulse Width at it's 50% height (PW50) gives:

$$PW50 = 2\sqrt{((f_x+d)(f_x+d+\delta)} \qquad \text{Equ. B.12}$$

Equations B.11 and B.12 may be reduced to even simpler forms by assuming near-zero magnetic media thickness:

$$e_x(X) \propto \frac{1}{1 + (x / (d + f_x))^2} \qquad \text{Equ. B.13}$$

$$PW50 = 2(d + f_x) \qquad \text{Equ. B.14}$$

Equation B.13 produces the so-called Lorentzian shape isolated pulse.

References for Appendix B.

MALLINSON, J.C., The Foundations of Magnetic Recording. Academic Press, Inc., USA, 1987.

MIDDLETON, B.K. Magnetic Recording, Vol. I: Technology. Chapter 5: Recording and Reproducing Process. Series Editors C. D. Mee, and E. D. Daniels, McGraw-Hill, Inc. USA, 1987.

Appendix C: Programme for Calculating Isolated Pulse Waveform Coefficients.

Written in FORTRAN-77 by Barry Good, Computing Dept, Polytechnic South West.

```
      PROGRAM CALCPULSE
      DOUBLE PRECISION A(9,9),B(9),C(9),X(9),Y(9),AA(9,9)
      DOUBLE PRECISION WKS1(9),WKS2(9),U
C     Data statements holding x,y coordinates of reference pulse
      DATA X/-5.47,-2.16,-1.34,-0.95,-0.67,-0.39,0.0,0.42,0.75/
      DATA Y/0.0,0.1,0.3,0.5,0.7,0.9,1.0,1.00,1.000/
      P(U)=(       ((((B(5)*U+B(4))*U+B(3))*U+B(2))*U+B(1))/
     /            ((((B(9)*U+B(8))*U+B(7))*U+B(6))*U+1.0D0) )
      DO 1 I=1,9
        A(I,1)=1.0
        A(I,2)=X(I)
        A(I,3)=X(I)**2
        A(I,4)=X(I)**3
        A(I,5)=X(I)**4
        A(I,6)=-Y(I)*A(I,2)
        A(I,7)=-Y(I)*A(I,3)
        A(I,8)=-Y(I)*A(I,4)
        A(I,9)=-Y(I)*A(I,5)
1       C(I)=Y(I)
      IFAIL=0
      CALL F04ATF(A,9,C,9,B,AA,9,WKS1,WKS2,IFAIL)
C     Print coefficients to screen
      PRINT *,'A(0) = ', B(1)
      PRINT *,'A(1) = ', B(2)
      PRINT *,'A(2) = ', B(3)
      PRINT *,'A(3) = ', B(4)
      PRINT *,'A(4) = ', B(5)
      PRINT *,'B(0) = ', 1.0D0
```

```
      PRINT *,'B(1) = ', B(6)
      PRINT *,'B(2) = ', B(7)
      PRINT *,'B(3) = ', B(8)
      PRINT *,'B(4) = ', B(9)
C     Generate points on isolated pulse for plotting and verification
      DO 2 I=-545,75,1
2       WRITE(*,1000)0.01D0*I,P(0.01D0*I)
1000    FORMAT(1X,2F10.3)
      STOP
      END
```

Appendix D: Polynomial Coefficients of Analytical Pulses.

This appendix lists the coefficients of the following equation used to generate the isolated pulses for the computer model.

$$f(t) = \frac{a[0] + a[1]t + a[2]t^2 + a[3]t^3 + a[4]t^4}{b[0] + b[1]t + b[2]t^2 + b[3]t^3 + b[4]t^4}$$

Inductive Head.

Left Hand Side Coefficients (-0.7mS to 0mS).

a[0] = 1.0
a[1] = 1.686037684194E-02
a[2] = 1.010845531056E-04
a[3] = 1.791685983396E-07
a[4] = 9.465082048366E-11

b[0] = 1.0
b[1] = 1.663340065266E-02
b[2] = 1.927853391487E-04
b[3] = 9.873698980734E-07
b[4] = 5.577926897161E-09

Right Hand Side Coefficients (0mS to 0.8mS).

a[0] = 1.0
a[1] = 1.069159470110E-02
a[2] = 7.919654258251E-06
a[3] = -6.665960569085E-08
a[4] = 4.802436072905E-11

b[0] = 1.0
b[1] = 1.047813788023E-02
b[2] = 1.017671231541E-04
b[3] = 1.612570906858E-07
b[4] = 7.889232628934E-10

Magneto-Resistive Head.

Left Hand Side Coefficients (-4.67mS to 0mS).

a[0] = 1.0
a[1] = 0.543152899120
a[2] = 2.014308045450
a[3] = 0.767498500152
a[4] = 7.519105275567E-02

b[0] = 1.0
b[1] = 0.538975094171
b[2] = 2.28096305343
b[3] = -0.178560580646
b[4] = 0.872452758027

Right Hand Side Coefficients (0mS to 1.0mS).

a[0] = 1.0
a[1] = -1.62558035337
a[2] = 0.875259954911
a[3] = -0.267186288244
a[4] = 3.260220221397E-02

b[0] = 1.0
b[1] = -1.58657485948
b[2] = 1.05682774379
b[3] = -0.202667007782
b[4] = -0.237394845510

Right Hand Side Coefficients (1.0mS to 4.24mS).

a[0] = 1.0
a[1] = -1.76363990836
a[2] = 0.746573193045
a[3] = -0.207425786074
a[4] = 2.749056807106E-02

b[0] = 1.0
b[1] = -1.71609518275
b[2] = 0.952224069788
b[3] = -0.216933245992
b[4] = -0.413199507672

Appendix E: Circuit Diagrams.

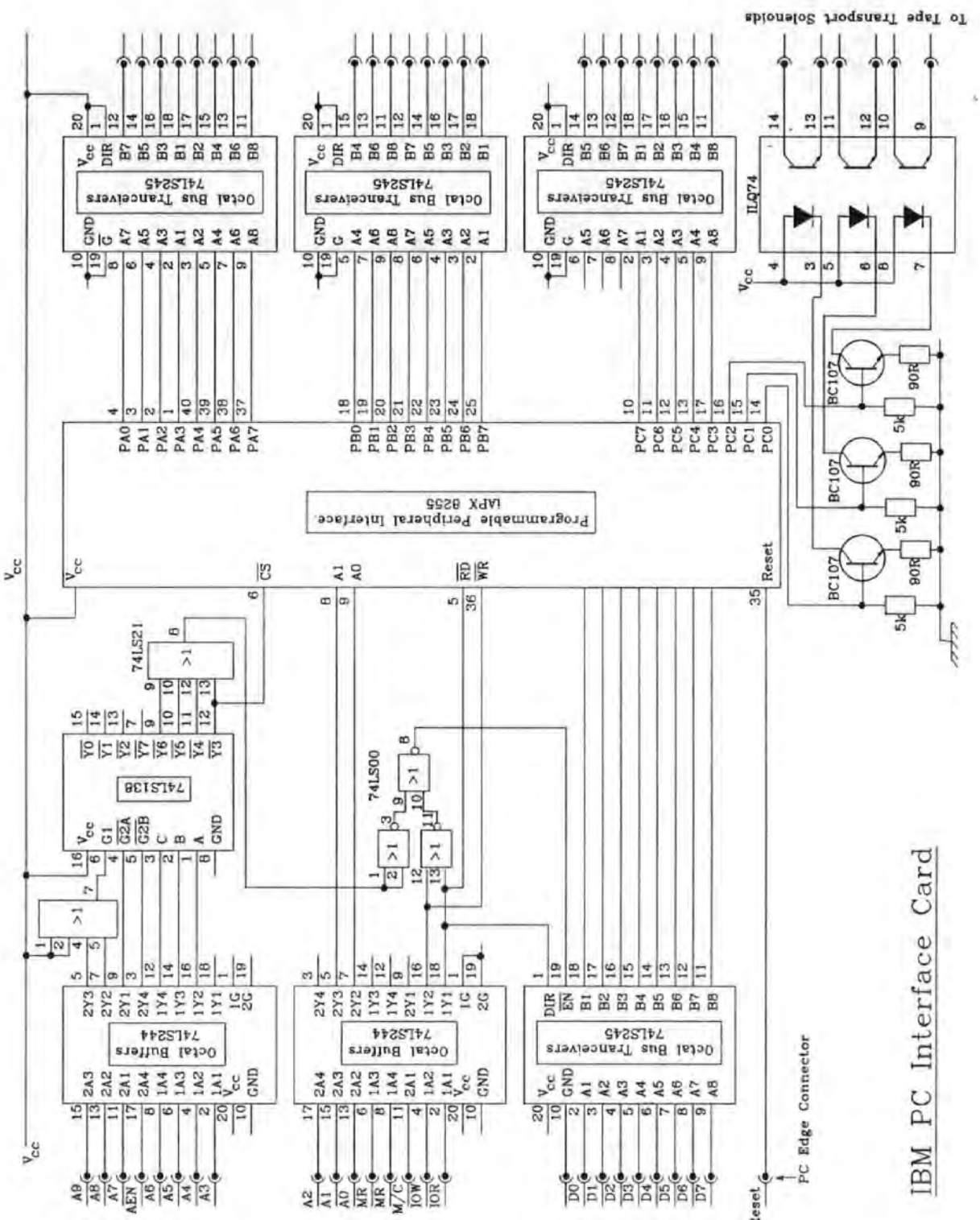


Fig. E.1. IBM PC Interface Card.

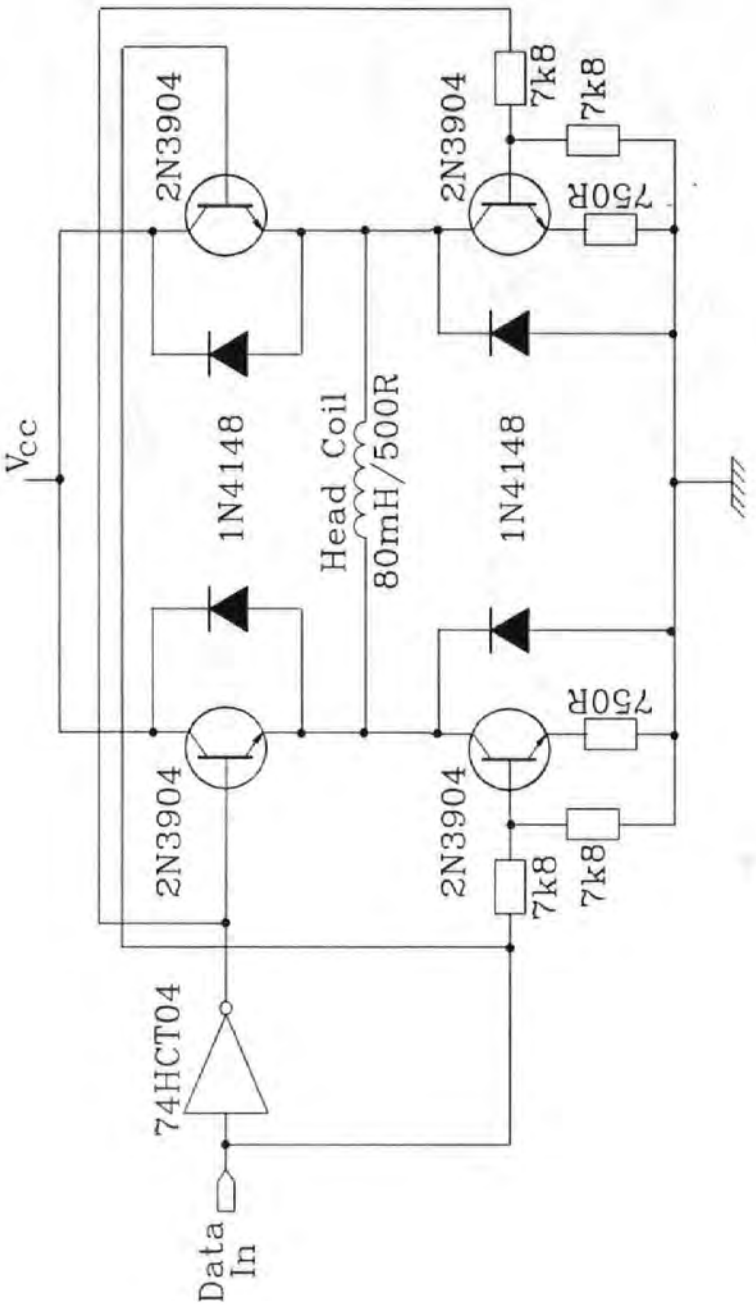


Fig. E.2. Write Amplifier Circuit (One Channel).

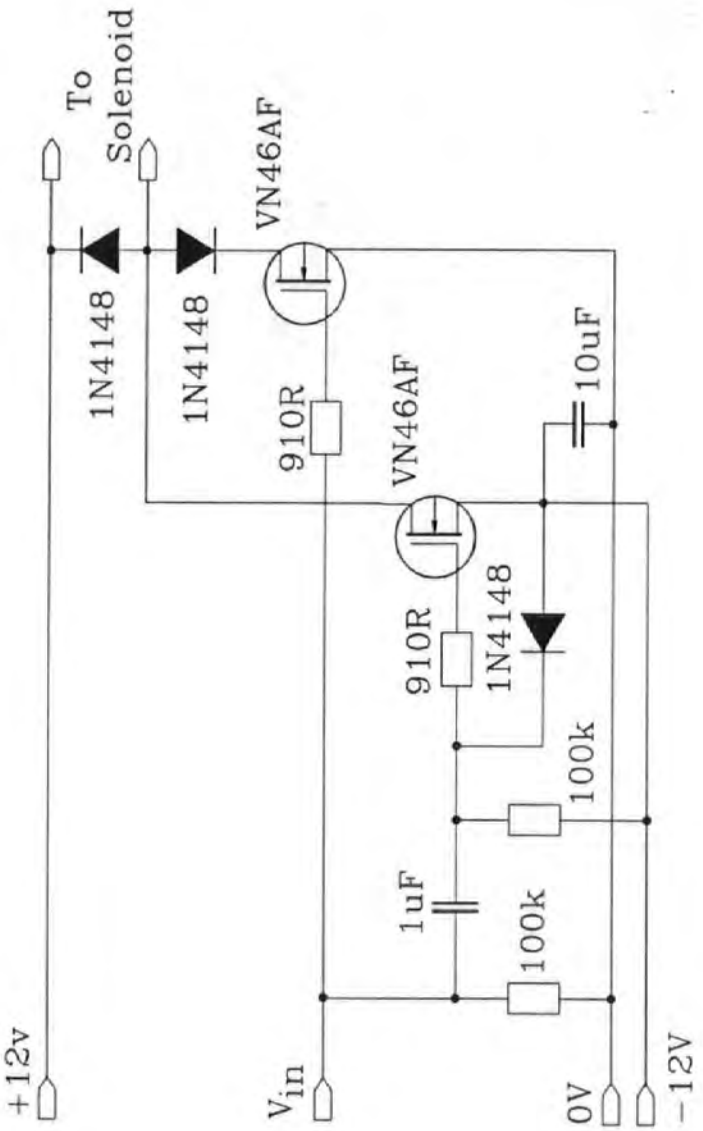


Fig. E.3. Solenoid Drive Circuit.

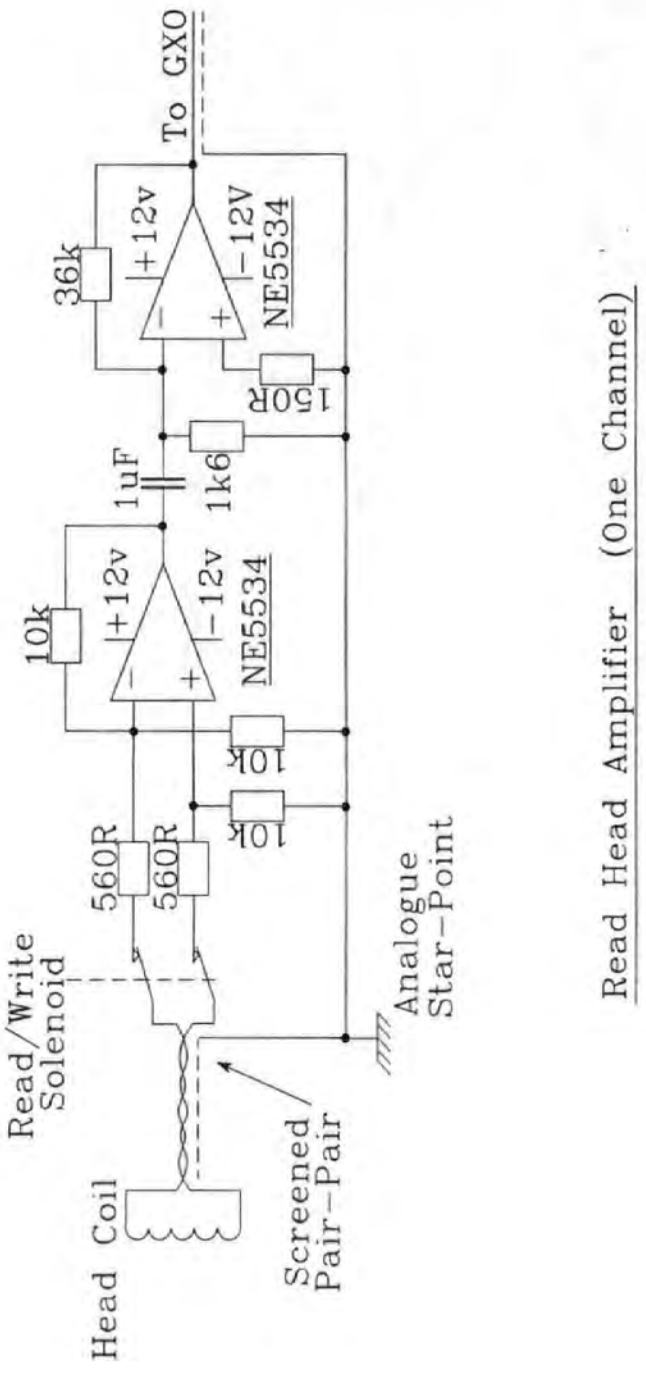
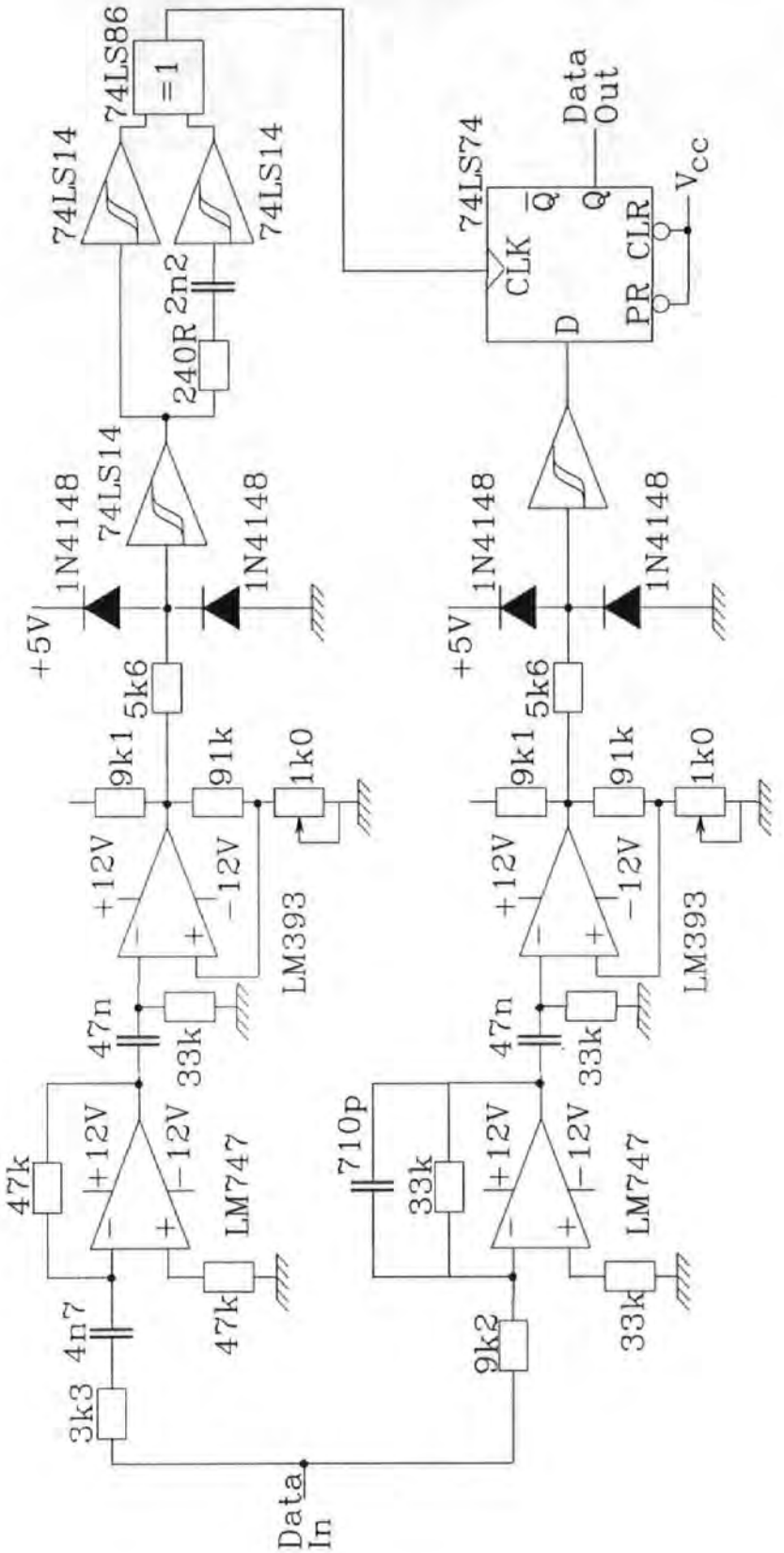


Fig. E.4. Read Head Amplifier Circuit (One Channel).



Gated Cross-Over Detector Circuit (One Channel).

Fig. E.5. Gated Cross-Over Circuit (One Channel).

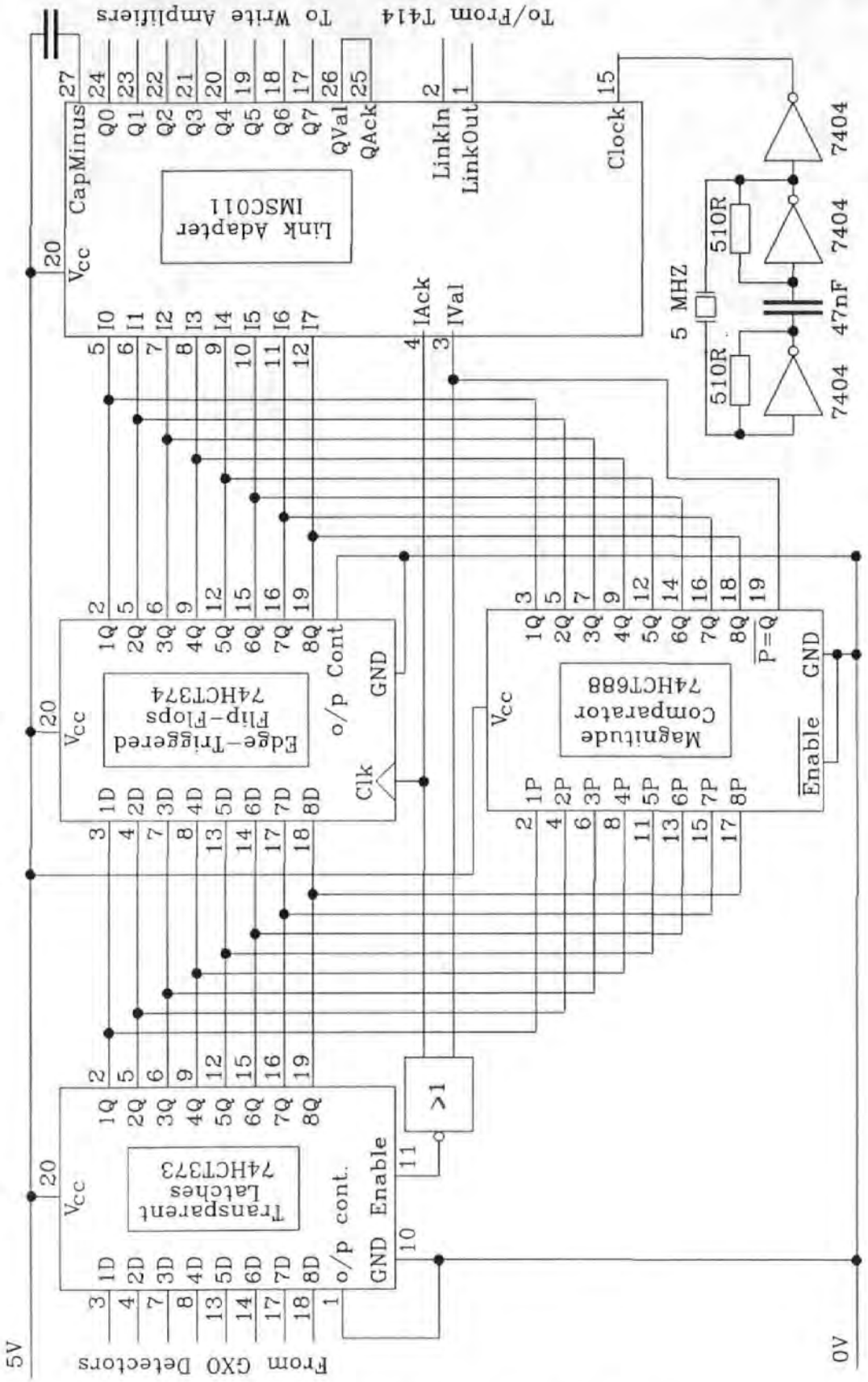


Fig. E.6. Link Adapter Interface Board.

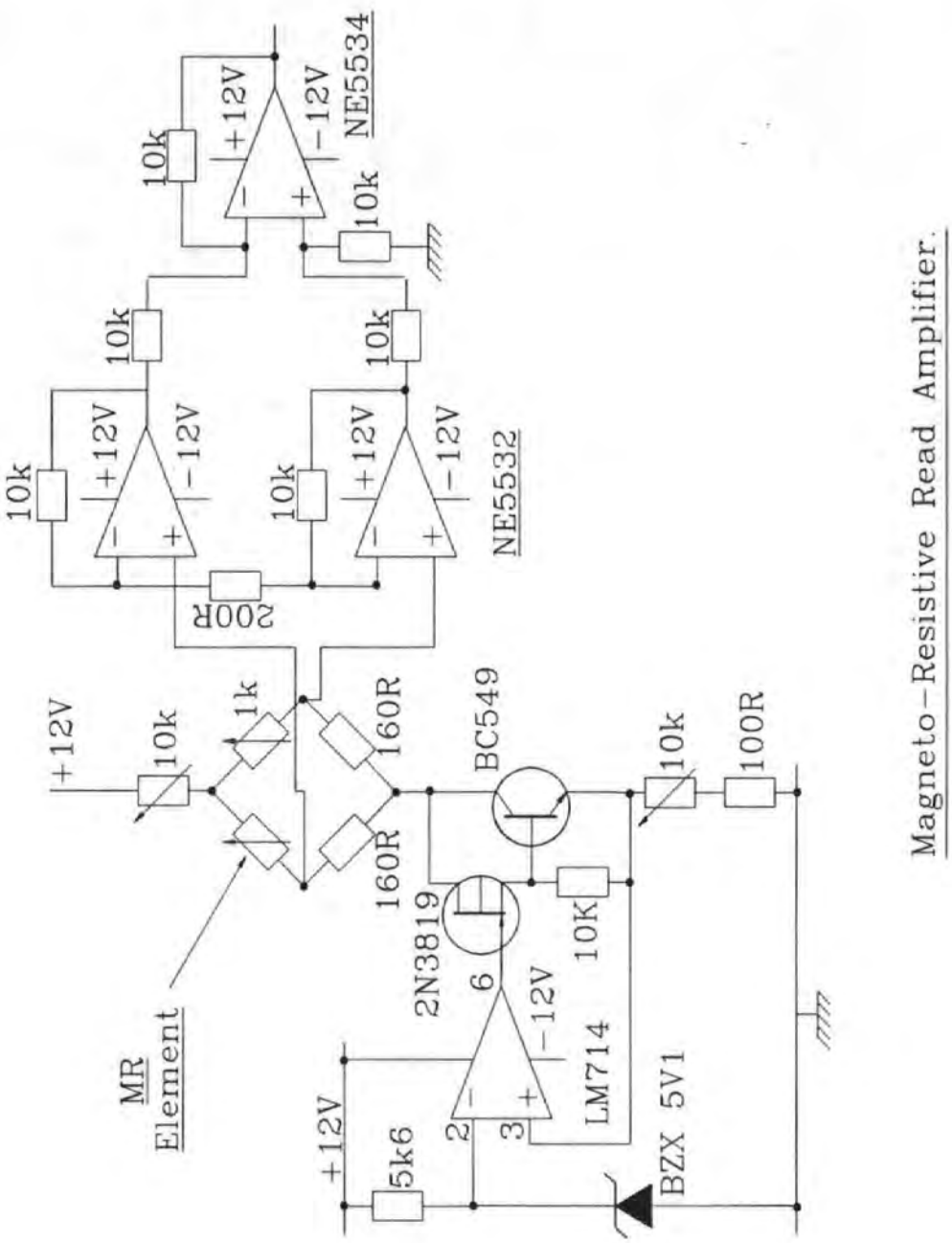


Fig. E.7. Magneto-Resistive Read Amplifier (One Channel).

Appendix F: Inductive Head Specification.

HQ551

Hard permalloy Head
4-Track, 4-channel

SPECIFICATIONS			
Impedance	1KHz 100μA	1000Ω±25%	
P.B. sensitivity	315Hz	−68.0dBm±2.0dB	
P.B. frequency response	10KHz/315Hz	+14.5±3.0dB	
	14KHz/315Hz	+12.0dB±4.0dB	
Bias current	60KHz peak over-1dB	350μA±25%	
Recording current	1KHz −10dB below saturation level	36μA±20%	
Rec. & P.B. sensitivity	1KHz	−66.5dBm±2.0dB	
Rec. & P.B. frequency response	10KHz/1KHz	−14.0dB±4.0dB	
Crosstalk (1KHz −30dBm)	1 ~ 2ch, 3 ~ 4ch	40dB min.	
	2 ~ 3ch	55dB min.	
	1 ~ 3ch, 2 ~ 4ch	50dB min.	
Differense of channel azimuth	10KHz	1dB max.	
Measuring tape	For P.B. frequency response For Rec. & P.B.	TEAC MTT-316 BASF QP-12	

Appendix G: *occam* Programme Listings.

The following ***occam*** code was that used in the real-time compact-cassette system, it's simulation, and the anti-displacement scheme. Much of the software was common to all three systems. Duplicated sections of code have been removed for compactness. Although the resultant no longer constitutes an ***occam*** programme, all of the relevent code is present (if disjointed).

The following process headings are highlighted in the text (in the following order), and indicate the start of significant process declarations. Also indicated are the systems the process was used in.

Compact Cassette	Model	LHD Comp Scheme	Code Heading
✓	✓	✓	Pseudo Random Binary Sequence Generator.
✓	✓	✓	Bi-Phase-L Channel Encoder.
✓			Timed Output.
	✓		Read Process (Linear Superposition).
	✓		Lateral Head Displacement Simulation.
	✓		Head Amplifier Model.
	✓	✓	Gated Cross Over Detector.
	✓	✓	Band-pass filter as Differentiator.
	✓	✓	Band-pass filter as gain block
	✓	✓	Comparator model with Hysteresis.
	✓	✓	Gaussian White Noise Generator.
	✓	✓	D-Type Flip-Flop.
	✓	✓	Sampled-to-Event Interface.
✓	✓	✓	Distribute Event Times and Data
✓	✓	✓	Bi-Phase-L Channel Decoder.
✓	✓	✓	PRBS Error Check and Classification.
✓	✓	✓	Draw Graphs.
✓	✓	✓	Prints Error Results onto Graph.
		✓	Anti-Displacement Compensation Scheme

```
{{{ SC genprbs
{{{F genprbs
{{{ Header
```

-- Pseudo Random Binary Sequence Generator.

```
-- Inputs Register Length and whether Data Staggered,
--   generates PRBS accordingly.
-- If no key pressed and not end of cassette, output next in series.
-- If key pressed, check for early Termination.
}}}
PROC gen.prbs (VAL INT reg.len, BOOL staggered,
               CHAN OF INT from.kb,
               CHAN OF ANY data.out)
 {{{ PROC defs
 {{{ SC init.pointers
 {{{F init.pointers
 {{{ Header
 -- Initialise Pointers.
 -- Internal to gen.prbs
 -- Sets up PRBS register pointers according to register length
 }}}
 PROC init.pointers (VAL INT reg.len, INT fb1, fb2, next.fb)
  SEQ
    next.fb := 0
    fb1 := (next.fb PLUS reg.len)
    {{{ set fb2
    CASE reg.len
      2
        fb2 := 1
      3
        fb2 := 2
      4
        fb2 := 3
      5
        fb2 := 3
      6
        fb2 := 5
      7
        fb2 := 6
      8
        fb2 := 6
      9
        fb2 := 5
      10
        fb2 := 7
      11
        fb2 := 9
      12
        fb2 := 11
      13
        fb2 := 12
      14
        fb2 := 13
      15
        fb2 := 14
    }}}
 :
 }}}F
 {{{ SC move.pointers
 {{{F move.pointers
 {{{ Header
 -- Move Pointers.
 -- Internal to gen.prbs
 -- Increments pointers to PRBS (as for Circular Buffer).
 }}}
 PROC move.pointers (INT ptr1, ptr2, ptr3)
  SEQ
    ptr1 := ( (ptr1 MINUS 1) /\ #0F)
    ptr2 := ( (ptr2 MINUS 1) /\ #0F)
    ptr3 := ( (ptr3 MINUS 1) /\ #0F)
 :
```

```
}}}F
}}}
}}}
{{{  decl's
{{{  #USE stuff
#USE pseudort :
#USE userio :
}}}
VAL run.time IS (50 TIMES (60 TIMES lp.tps) ) :-- ie 50 minutes record time MAX
BOOL running :
[16]INT s.reg :
INT fb1, fb2, next.fb :
INT start.time, char:
TIMER clock :
}}}
SEQ
  {{{  init
  clock ? start.time
  running:= TRUE
  init.pointers (reg.len, fb1, fb2, next.fb)
  {{{  fill s.reg
  IF  -- Fill PRBS register depending on whether data staggered or not
    staggered
      {{{
      CASE reg.len
        2
          [s.reg FROM 1 FOR 2] := [#0D, #0B]
        3
          [s.reg FROM 1 FOR 3] := [#09, #03, #07]
        4
          [s.reg FROM 1 FOR 4] := [#01, #03, #07, #0F]
        5
          [s.reg FROM 1 FOR 5] := [#01, #03, #07, #0F, #0F]
      }}}
    NOT staggered
      SEQ i = 0 FOR 16
        s.reg [i] := #0F
  }}}
  }}}
  WHILE running
    PRI ALT
      from.kb ? char
        {{{  process
        IF
          char <> stopchar
            SKIP
          TRUE -- Early Termination, Start shut down mechanism
            running := FALSE
        }}}
      clock ? AFTER (start.time PLUS run.time)
        running := FALSE
      TRUE & SKIP
        {{{  clock PRBS
        SEQ
          s.reg[next.fb] := s.reg[fb1] >< s.reg[fb2] -- calculate new PRBS datum
          data.out ! s.reg[next.fb]                  -- output new datum
          move.pointers (fb1, fb2, next.fb)          -- clock PRBS
        }}}
  data.out ! terminate -- Last instruction, pass on terminate
:
}}}F
}}}
{{{  SC man.code
{{{F man.code
{{{  Header
-- Bi-Phase-L Channel Encoder.

}}}
PROC man.code (CHAN OF INT data.in, data.out)
  {{{  decl's
  #USE pseudort :
```

```
  INT data :
  BOOL running :
  }}}
  SEQ
    {{{  initialise
    running := TRUE
    }}}
    WHILE running
      SEQ
        data.in ? data
        IF
          data <> terminate
            {{{  process
            SEQ -- channel code datum
              data.out ! ((~data) /\ #0F) -- inverted
              data.out ! (data /\ #0F)
            }}}
          TRUE
            {{{  finish
            SEQ
              running := FALSE
            }}}
      data.out ! terminate -- Last instruction, pass on terminate
:
}}}F
}}}
{{{  SC timed.output
{{{F timed.output
{{{  Header
```

-- Timed Output.

```
-- Receives data from channel encoding process, outputs at timed interval
-- according to data rate. Must be run at High Priority for accuracey.
}}}
PROC timed.output (VAL INT data.bps,
                   CHAN OF INT data.in,
                   CHAN OF ANY to.screen,
                   CHAN OF BYTE data.out)
  {{{  decl's
  #USE userio :
  #USE pseudort :
  VAL code.bit.time IS ((hp.tps/data.bps) >> 1) :
  TIMER clock :
  BOOL running :
  INT data, time :
  }}}
  SEQ
    {{{  initialise
    running := TRUE
    clock ? time
    time := (time PLUS (code.bit.time TIMES 10) ) -- allow time for rest of code
    }}}
    WHILE running
      SEQ
        data.in ? data
        IF
          data <> terminate
            {{{  output data
            SEQ
              clock ? AFTER time  -- wait till time for next datum output
              data.out ! BYTE data  -- output data
              time := (time PLUS code.bit.time)  - calculate time of next datum output
            }}}
          TRUE
            running := FALSE
:
}}}F
}}}
{{{  decl's
#USE userio :
#USE mech :
```

```
#USE pseudort :
BOOL data.valid, staggered :
INT data.bps, reg.len, char :
}}}
SEQ
 {{{ Get Valid data for this run
 data.valid := FALSE
 WHILE (NOT data.valid)
  SEQ
   {{{ get values for this run
   newline (screen)
   write.full.string (screen, "Input Data Freq (bps) ")
   read.echo.char (keyboard, screen, char)
   read.echo.int (keyboard, screen, data.bps, char)
   newline (screen)
   write.full.string (screen, "Input Register Length ")
   read.echo.char (keyboard, screen, char)
   read.echo.int (keyboard, screen, reg.len, char)
   newline (screen)
   write.full.string (screen, "Staggered (s) or Unstaggered (u) ")
   read.echo.char (keyboard, screen, char)
   IF
    char = (INT 's')
     staggered := TRUE
    TRUE
     staggered := FALSE
   newline (screen)
   newline (screen)

   }}}
   {{{ print out confirmation
   write.full.string (screen, "Data Freq ")
   write.int (screen, data.bps, 6)
   newline (screen)
   write.full.string (screen, "Register Length ")
   write.int (screen, reg.len, 4)
   newline (screen)
   IF
    staggered
     write.full.string (screen, "Data Staggered. ")
    TRUE
     write.full.string (screen, "Data NOT Staggered. ")
   newline (screen)
   newline (screen)
   newline (screen)
   write.full.string (screen, "          Is this correct ? (y/n) ")
   read.echo.char (keyboard, screen, char)
   IF
    char = (INT 'y')
     data.valid := TRUE
    TRUE
     SKIP
   }}}
 newline (screen)
 newline (screen)
 newline (screen)
 }}}
 {{{ Initialse
 mech.init (screen) -- Send Interupt to PC Initialising 8255 PIA
 mech.menu (screen) -- Send Interupt to PC to Display Tape Transport Menu
 newline (screen)
 set.clocks (0)    -- Zero transputer clock (avoids timer 'roll-over')
 }}}
 {{{ text stuff
 write.full.string (screen, "          Mechanism MUST BE IN WRITE MODE ")
 newline (screen)
 newline (screen)
 write.full.string (screen, "Press Q to Quit, any other to continue...")
 read.char (keyboard, char)
 newline (screen)
 newline (screen)
 }}}
```

```
IF
  char < > (INT 'q')
    {{{  start recording
    SEQ
      write.full.string (screen, "Recording Data.....")
      CHAN OF ANY from.code, from.gen :
      CHAN OF BYTE to.link :
      PLACE to.link AT linkout1 :
```

-- MAIN PROGRAMME CONSTRUCT FOR WRITING.

```
      PRI PAR
        timed.output (data.bps, from.code, screen, to.link)
        PAR
          gen.prbs (reg.len, staggered, keyboard, from.gen)
          man.code (from.gen, from.code)
      mech.stop (screen) -- When finished STOP tape mechanism
      newline (screen)
    }}}
  TRUE
    SKIP
write.full.string (screen, "All done. Press any key to return to TDS...")
read.char (keyboard, char)
newline (screen)
```

Start of Simulation Code

```
{{{  decl's
{{{  #USE stuff
#USE pseudort :
#USE userio :
#USE interf :
#USE userhdr :
#USE strings :
#USE t4math :
}}}
CHAN OF INT from.param.fold :
VAL INT top.fold IS 1 :
INT char, param.fold.result, start.time :
TIMER clock :
}}}
SEQ
  {{{  start message
  clock ? start.time
  write.text.line (screen, "                    Started....")
  }}}
  PAR
    {{{  programme,
    BOOL more.runs :
    SEQ
      more.runs := TRUE
      WHILE more.runs
        PRI ALT
          keyboard ? char
            {{{
            IF
              char < > stopchar
                SKIP
              TRUE
                more.runs := FALSE
            }}}
          more.runs & SKIP
            {{{  run programme
            {{{  local decl's "LOG" channel
            INT log.fold.result, log.fold.num :
            CHAN OF ANY to.log :
            }}}
            PAR
              {{{  run test
              {{{  decl's
              {{{  CHAN decl's
              {{{  PROTOCOL DEF --    Other Definitions of 'INT.OR.FLOAT' removed.
```

```
PROTOCOL INT.OR.FLOAT
  CASE
    int ; INT
    float ; REAL32
:
}}}
{{{  CHAN decl's
[num.tracks]CHAN OF INT from.prbs, from.disp, from.read :
[num.tracks][2]CHAN OF ANY from.headamp :
[num.tracks]CHAN OF INT from.gxo :
CHAN OF INT stop.prbs :
}}}
}}}
{{{  define maximun values for memory allocation
VAL REAL32 T IS (one / fs) :

VAL INT max.block.size IS 300000 :
VAL INT max.data.rate IS 10000 :
VAL INT min.data.rate IS 1000 :
VAL INT range IS (INT ROUND ((max.neg + max.pos) / T)) :
VAL INT max.pulse.sep IS (INT ROUND
                    ((one / (two * (REAL32 ROUND min.data.rate))) / T)) :
VAL INT max.prbs.reg.len IS 3 :
VAL INT max.prbs.seq.len IS 7 :
VAL INT max.snapshot.len IS 500 :
VAL INT max.his.len IS (max.prbs.seq.len PLUS 1) :
VAL INT max.ref.size IS 8 :

}}}
{{{  memory allocation
[num.tracks][max.snapshot.len]REAL32 pre.headamp, post.headamp :
[num.tracks][max.snapshot.len]REAL32 gate.ana, pol.ana :
[num.tracks][max.snapshot.len]BYTE gate.dig, pol.dig, gxo.dig :
[num.tracks][range PLUS 1]REAL32 basic.pulse :
[max.block.size]BYTE data :
[max.block.size]INT times :
[num.tracks][max.his.len]INT burst.his :
[num.tracks]INT count, class.good, class.bad :
[num.tracks]INT good.bits, bad.bits, lost.synch :
[num.tracks]REAL32 track.rate, skew.bits :
[num.tracks]INT skew.samples :
REAL32 rate :
[100]BYTE comment.text :
[max.ref.size]BYTE ref.date.text :
REAL32 gate.threshold, pol.threshold, pol.noise.pp, gate.noise.pp :
REAL32 bounce.pp, displacement, write.width, read.width, side.write.width :
INT comment.len, ref.date.len, data.rate, sim.data.rate :
INT char, fold.num, result, max.bad, min.good :
INT block.size, prbs.reg.len, his.len :
INT waveform.name.len :
[abs.id.size]BYTE waveform.filename :
BOOL staggered :
}}}
}}}
{{{  PROC decl's
{{{  SC get.parameters
{{{F get.parameters
{{{  PROC get.parameters ( PARAMETER LIST )
PROC get.parameters (CHAN OF INT data.in, CHAN OF ANY echo.out,
                     []BYTE waveform.filename, INT waveform.name.len,
                     INT data.rate,
                     REAL32 read.width, write.width, side.write.width,
                     REAL32 gate.threshold, pol.threshold, displacement,
                     []REAL32 skew.bits, BOOL staggered,
                     INT max.bad, min.good, prbs.reg.len, INT block.size,
                     []BYTE comment.text, INT comment.len, BOOL more.runs)
}}}
  {{{  decl's
  #USE pseudort :
  #USE strings :
  #USE userio :
  [40]BYTE text :
```

```
INT char, text.len :
}}}
SEQ
  {{{  get waveform filename
  read.echo.text.line (data.in, echo.out, waveform.name.len,
                       waveform.filename, char)
  waveform.name.len := (waveform.name.len MINUS 1) -- remove *c
  }}}
  {{{  get data freq
  read.echo.char (data.in, echo.out, char)
  read.echo.int (data.in, echo.out, data.rate, char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get read width
  read.echo.char (data.in, echo.out, char)
  read.echo.real32 (data.in, echo.out, read.width, char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get write width
  read.echo.char (data.in, echo.out, char)
  read.echo.real32 (data.in, echo.out, write.width, char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get side write width
  read.echo.char (data.in, echo.out, char)
  read.echo.real32 (data.in, echo.out, side.write.width, char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get gate comp thresh
  read.echo.char (data.in, echo.out, char)
  read.echo.real32 (data.in, echo.out, gate.threshold, char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get xover comp thresh
  read.echo.char (data.in, echo.out, char)
  read.echo.real32 (data.in, echo.out, pol.threshold, char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get track disp
  read.echo.char (data.in, echo.out, char)
  read.echo.real32 (data.in, echo.out, displacement, char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get skew
  read.echo.char (data.in, echo.out, char)
  SEQ track = 0 FOR num.tracks
    read.echo.real32 (data.in, echo.out, skew.bits[track], char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get un/staggered
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  text.len := (text.len MINUS 1) -- remove *c
  VAL []BYTE stagger.state IS "staggered        " :
  IF
    {{{  staggered
    eqstr ([text FROM 0 FOR text.len], [stagger.state FROM 0 FOR text.len])
    }}}
      staggered := TRUE
    TRUE
      staggered := FALSE
  }}}
  {{{  get max bad
  read.echo.char (data.in, echo.out, char)
  read.echo.int (data.in, echo.out, max.bad, char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get min good
  read.echo.char (data.in, echo.out, char)
  read.echo.int (data.in, echo.out, min.good, char)
  read.echo.text.line (data.in, echo.out, text.len, text, char)
  }}}
  {{{  get reg len
```

```
    read.echo.char (data.in, echo.out, char)
    read.echo.int (data.in, echo.out, prbs.reg.len, char)
    read.echo.text.line (data.in, echo.out, text.len, text, char)
    }}}
    {{{  get block size
    read.echo.char (data.in, echo.out, char)
    read.echo.int (data.in, echo.out, block.size, char)
    read.echo.text.line (data.in, echo.out, text.len, text, char)
    }}}
    {{{  get comment
    read.echo.text.line (data.in, echo.out, comment.len, comment.text, char)
    }}}
    {{{  get terminate or not
    read.echo.text.line (data.in, echo.out, text.len, text, char)
    text.len := (text.len MINUS 1) -- remove *c
    VAL []BYTE last.test IS "last test                    " :
    IF
      {{{  last run
      eqstr ([text FROM 0 FOR text.len], [last.test FROM 0 FOR text.len])
      }}}
        more.runs := FALSE
      TRUE
        SKIP
    }}}
:
}}}F
}}}
{{{  SC print.vals
{{{F print.vals
PROC print.vals (VAL REAL32 T,
                 VAL INT pulse.sep, snapshot.len, sim.data.rate,
                 CHAN OF ANY data.out)
  #USE userio :
  SEQ
    {{{  print pulse.sep
    write.full.string (data.out, "Pulse Spacing ")
    write.int (data.out, pulse.sep, 0)
    newline (data.out)
    }}}
    {{{  print T
    write.full.string (data.out, "T ")
    write.real32 (data.out, T, 0, 0)
    newline (data.out)
    }}}
    {{{  print snapshot.len
    write.full.string (data.out, "snapshot.len ")
    write.int (data.out, snapshot.len, 0)
    newline (data.out)
    }}}
    {{{  print sim.data.rate
    write.full.string (data.out, "sim.data.rate")
    write.int (data.out, sim.data.rate, 0)
    newline (data.out)
    }}}
:
}}}F
}}}
{{{  SC calc.rates
{{{F calc.rates
PROC calc.rates (VAL []INT class.good, class.bad, lost.synch,
                 VAL INT max.bad, block.size,
                 []REAL32 track.rate,
                 REAL32 rate)
  #USE pseudort :
  VAL REAL32 min.error.rate IS (one / (REAL32 ROUND block.size)) :
  VAL INT num.tracks IS (SIZE class.good) :
  SEQ
    {{{  calc rate for each track
    SEQ track = 0 FOR num.tracks
      VAL INT total.bits IS ((class.good[track] PLUS class.bad[track]) PLUS
                             (lost.synch[track] TIMES (max.bad PLUS 1))) :
      IF
```

```
      total.bits > 0
        {{{
        VAL REAL32 error.rate IS ((REAL32 ROUND (total.bits -
                    class.good[track])) / (REAL32 ROUND total.bits)) :
        IF
          error.rate > min.error.rate
            track.rate[track] := error.rate
          TRUE
            track.rate[track] := min.error.rate
        }}}
      TRUE
        track.rate[track] := one
    }}}
    {{{  calc overall rate
    rate := zero
    SEQ track = 0 FOR num.tracks
      rate := (rate + track.rate[track])
    rate := (rate / (REAL32 ROUND num.tracks))
    }}}
:
}}}F
}}}
{{{  SC print.totals
{{{F print.totals
PROC print.totals (VAL []INT count, lost.synch, class.good, class.bad,
                   VAL []INT good.bits, bad.bits, [][]INT burst.his,
                   VAL INT his.len, VAL []REAL32 track.rate, VAL REAL32 rate,
                   CHAN OF ANY data.out)
  #USE userio :
  VAL INT num.tracks IS (SIZE count) :
  SEQ
    SEQ track = 0 FOR num.tracks
      SEQ
        {{{  track
        write.full.string (data.out, "Track ")
        write.int (data.out, (track PLUS 1), 0)
        newline (data.out)
        }}}
        {{{  count and lost synch
        write.full.string (data.out, "Count          ")
        write.int (data.out, count[track], 0)
        write.full.string (data.out, "      Lost Synch      ")
        write.int (data.out, lost.synch[track], 0)
        newline (data.out)
        }}}
        {{{  class good and class bad
        write.full.string (data.out, "Class Good      ")
        write.int (data.out, class.good[track], 0)
        write.full.string (data.out, "      Class Bad       ")
        write.int (data.out, class.bad[track], 0)
        newline (data.out)
        }}}
        {{{  good bits and bad bits
        write.full.string (data.out, "Good Bits       ")
        write.int (data.out, good.bits[track], 0)
        write.full.string (data.out, "      Bad Bits        ")
        write.int (data.out, bad.bits[track], 0)
        newline (data.out)
        }}}
        {{{  burst history
        write.full.string (data.out, "Burst History   ")
        SEQ b.his = 1 FOR (his.len MINUS 1)
          SEQ
            write.int (data.out, burst.his[track][b.his], 4)
            write.full.string (data.out, " ")
        newline (data.out)
        }}}
        {{{  track rate
        write.full.string (data.out, "Track Rate ")
        write.real32 (data.out, track.rate[track], 1, 5)
        newline (data.out)
        }}}
```

```
    {{{  overall rate
    write.full.string (data.out, "Overall Rate ")
    write.real32 (data.out, rate, 0, 5)
    newline (data.out)
    }}}
:
}}}F
}}}
{{{  SC print.elapsed.time
{{{F print.elapsed.time
PROC print.elapsed.time (CHAN OF ANY data.out, VAL INT start.time)
  #USE userio :
  #USE pseudort :
  INT elapsed, time.now, minutes, seconds :
  TIMER clock :
  SEQ
    clock ? time.now
    elapsed := ((time.now MINUS start.time) / lp.tps)
    minutes := (elapsed / 60)
    seconds := (elapsed REM 60)
    write.full.string (data.out, "Elapsed time ")
    write.int (data.out, minutes, 0)
    write.full.string (data.out, " mins ")
    write.int (data.out, seconds, 0)
    write.full.string (data.out, " secs. ")
:
}}}F
}}}
}}}
SEQ
  {{{  get next set of parameters from fold
  {{{  message
  print.elapsed.time (to.log, start.time)
  write.text.line (to.log, "            Read parameters...")
  }}}
  get.parameters (from.param.fold, to.log,
                  waveform.filename, waveform.name.len,
                  data.rate, read.width, write.width, side.write.width,
                  gate.threshold, pol.threshold, displacement, skew.bits,
                  staggered, max.bad, min.good, prbs.reg.len, block.size,
                  comment.text, comment.len, more.runs)
  {{{  message
  print.elapsed.time (to.log, start.time)
  write.text.line (to.log, "                Parameters read")
  }}}
  }}}
  {{{  set VAL's for this run
  VAL REAL32 gate.stan.dev IS 0.00186(REAL32) :
  VAL REAL32 pol.stan.dev IS  0.00041(REAL32) :
  VAL REAL32 pulse.sep.t IS (one / (two * (REAL32 ROUND data.rate))) :
  VAL INT pulse.sep IS (INT ROUND (pulse.sep.t / T)) :

  VAL INT his.len IS (max.bad PLUS 1) :
  VAL INT prbs.seq.len IS INT ROUND
                    ((POWER (two, (REAL32 ROUND prbs.reg.len))) - one) :
  VAL INT complete.snapshot.len IS (INT ROUND ((REAL32 ROUND prbs.seq.len) *
                          ((2.5(REAL32)) * (REAL32 ROUND pulse.sep)))) :
  VAL INT snapshot.step.size IS (1 PLUS (INT TRUNC ((REAL32 ROUND
            complete.snapshot.len) / (REAL32 ROUND max.snapshot.len)))) :
  VAL INT snapshot.len IS (complete.snapshot.len / snapshot.step.size) :
  VAL INT settle.time IS (2 TIMES range) :
  }}}
  SEQ
    {{{  print val's out
    sim.data.rate := (INT ROUND (one / (two * ((REAL32 ROUND pulse.sep) * T))))
    print.vals (T, pulse.sep, snapshot.len, sim.data.rate, to.log)
    }}}
    {{{  convert 'skew.bits' to 'skew.samples'
    VAL REAL32 bits.2.samples IS (T * (REAL32 ROUND sim.data.rate)) :
    SEQ track = 0 FOR num.tracks
      skew.samples[track] := (INT ROUND (skew.bits[track] / bits.2.samples))
    }}}
```

```
{{{F fill data and time arrays
{{{  message
print.elapsed.time (to.log, start.time)
write.text.line (to.log, "        Filling array with data....")
}}}
{{{  PROC decl's

-- Code for gen.prbs and manch.encode removed.
-- Refer to Appendix XXX for their declaration.

{{{  SC read
{{{F read
{{{  Header
```

- Read Process. Main Super-Position Process.

```
-- Reads digital data, outputs analogue data representing replay signal.
-- Pulse Seperation defines data rate in terms of samples between Pulses.
-- Skew is amount of Data Skew.
}}}
PROC read (CHAN OF INT data.in, []REAL32 pulse, VAL INT pulse.sep,
          VAL INT skew, CHAN OF INT.OR.FLOAT data.out)
  {{{  SC mac.pulse
  {{{F mac.pulse
  {{{  Header
  -- Function calculates pulse shape chosen by Mackintosh.
  -- Internal to read process. Returns one REAL32.
  }}}
  REAL32 FUNCTION mac.pulse (VAL REAL32 x)
    #USE pseudort :
    VAL REAL32 squared IS (x * x) :
    VALOF
      SEQ
        SKIP
      RESULT (one/(one + (squared + (squared * squared))))
  :
  }}}F
  }}}
  {{{  SC gen.gauss
  {{{F gen.gauss
  {{{  Header
  -- Generates Gaussian Noise.
  -- Receives Standard Deviation, Arithmetic Mean and last number in sequence.
  }}}
  PROC gen.gauss (INT32 seed,
                  VAL REAL32 stan.dev, mean,
                  REAL32 norm.num)
    #USE t4math :
    #USE pseudort :
    VAL INT k IS 12 : --number of random numbers summed to produce Normal Dist
    VAL REAL32 twelve IS 12.0(REAL32) :
    VAL REAL32 shift IS ((REAL32 ROUND k) / two) :
    -- VAL REAL32 divisor IS SQRT ((REAL32 ROUND k) / twelve) : -- when k <> 12
    SEQ
      norm.num := zero
      SEQ j = 0 FOR k
        REAL32 temp :
        SEQ
          temp, seed := RAN (seed)
          norm.num := norm.num + temp
      norm.num := (((norm.num - shift) * stan.dev) + mean)
      -- norm.num := ((((norm.num - shift) / divisor) * stan.dev) + mean)
  :
  }}}F
  }}}
  {{{  SC build.ind.pulse
  {{{F build.ind.pulse
  {{{  Header
  -- Construct Inductive Head Isolated Pulse.
  -- Internal to read process.
  -- Receives data array that is filled with REAL32 data.
  }}}
```

```
#USE pseudort :
PROC build.ind.pulse ([]REAL32 data)
 {{{ PROC's
 {{{F nag.left.pulse
 {{{ Header
 -- Function calculates Value of curve at x, Left Side of Pulse.
 -- Internal to build.ind.pulse
 }}}
 REAL32 FUNCTION nag.left.pulse (VAL REAL32 x)
  VAL INT order IS 4 : -- Order of Equations used
  {{{ VAL a.coef's
  VAL []REAL64 a.coeff IS [ 1.0(REAL64),
                    1.6860376841 94E-02(REAL64),
                    1.0108455310 56E-04(REAL64),
                    1.7916859833 96E-07(REAL64),
                    9.4650820483 66E-11(REAL64) ] :
  }}}
  {{{ VAL b.coef's
  VAL []REAL64 b.coeff IS [ 1.0(REAL64),
                    1.6633400652 66E-02(REAL64),
                    1.9278533914 87E-04(REAL64),
                    9.8736989807 34E-07(REAL64),
                    5.5779268971 61E-09(REAL64) ] :
  }}}
  VAL REAL64 microsecond IS 1.0E-6(REAL64) :
  VAL REAL64 x64 IS ((REAL64 ROUND x) / microsecond):
  REAL64 numer, denom, x.power :
  VALOF
   SEQ
    x.power := x64
    numer := a.coeff[0]
    denom := b.coeff[0]
    SEQ i = 1 FOR order -- main loop
     SEQ
      numer := numer + (a.coeff[i] * x.power)
      denom := denom + (b.coeff[i] * x.power)
      x.power := x.power * x64
   RESULT (REAL32 ROUND (numer / denom))
:
}}}F
{{{F nag.right.pulse
{{{ Header
-- Function calculates Value of curve at x, Right Side of Pulse.
-- Internal to build.ind.pulse
}}}
REAL32 FUNCTION nag.right.pulse (VAL REAL32 x)
  VAL INT order IS 4 :
  {{{ VAL a.coef's
  VAL []REAL64 a.coeff IS [ 1.0(REAL64),
                    1.0691594701 10E-02(REAL64),
                    7.9196542582 51E-06(REAL64),
                   -6.6659605690 85E-08(REAL64),
                    4.8024360729 05E-11(REAL64) ] :
  }}}
  {{{ VAL b.coef's
  VAL []REAL64 b.coeff IS [ 1.0(REAL64),
                    1.0478137880 23E-02(REAL64),
                    1.0176712315 41E-04(REAL64),
                    1.6125709068 58E-07(REAL64),
                    7.8892326289 34E-10(REAL64) ] :
  }}}
  VAL REAL64 microsecond IS 1.0E-6(REAL64) :
  VAL REAL64 x64 IS ((REAL64 ROUND x) / microsecond):
  REAL64 numer, denom, x.power :
  VALOF
   SEQ
    x.power := x64
    numer := a.coeff[0]
    denom := b.coeff[0]
    SEQ i = 1 FOR order
     SEQ
      numer := numer + (a.coeff[i] * x.power)
```

```
        denom := denom + (b.coeff[i] * x.power)
        x.power := x.power * x64
    RESULT (REAL32 ROUND (numer / denom))
:
}}}F
}}}
{{{  decl's
VAL INT series.len IS (SIZE data) :
VAL INT pulse.len IS (series.len / 1) :
VAL REAL32 head.amp.gain IS 357.0(REAL32) : -- Used to correctly scale pulse
VAL REAL32 max.ampl IS (0.625(REAL32) / head.amp.gain) : -- Used to scale pulse
VAL REAL32 scale IS 0.9507(REAL32) :-- ensures filtered pulse correct height
VAL REAL32 range IS (max.pos + max.neg) :
VAL REAL32 sample.width IS (range / (REAL32 ROUND (pulse.len MINUS 1))) :
}}}
SEQ
  {{{  build negative half
  SEQ i = 0 FOR ((INT ROUND (max.neg / sample.width)) PLUS 1)
    VAL REAL32 x IS (((REAL32 ROUND i) * sample.width) - max.neg) :
    data[i] := ((neg.left.pulse(x) / scale) * max.ampl)
  }}}
  {{{  build right half
  SEQ i = (INT ROUND (max.neg / sample.width)) FOR
                        ((INT ROUND (max.pos / sample.width)) PLUS 1)
    VAL REAL32 x IS (((REAL32 ROUND (i PLUS 1)) * sample.width) - max.neg):
    data[i] := ((neg.right.pulse(x) / scale) * max.ampl)
  }}}
:
}}}F
}}}
{{{  SC build.mr.pulse
{{{F build.mr.pulse
{{{  Header
-- Build Magneto-Resistive Pulse.
-- Internal to build.mr.pulse.
-- In three sections, one for Left, two for Right Hand Side.
}}}
#USE pseudort :
PROC build.mr.pulse ([]REAL32 data)
  {{{  PROC's
  {{{F neg.left.pulse
  {{{  Header
  -- Calculate value of Left hand side of pulse.
  -- Internal to build.mr.pulse.
  }}}
  REAL32 FUNCTION neg.left.pulse (VAL REAL32 x)
    VAL INT order IS 4 : -- Order of Equation
    {{{  VAL a.coef's
    VAL []REAL64 a.coeff IS [ 1.0(REAL64),
                              0.543152899120(REAL64),
                              2.014308045450(REAL64),
                              0.787498500152(REAL64),
                              7.519105275567E-02(REAL64) ] :
    }}}
    {{{  VAL b.coef's
    VAL []REAL64 b.coeff IS [ 1.0(REAL64),
                              0.538975094171(REAL64),
                              2.28096305343(REAL64),
                             -0.178560580646(REAL64),
                              0.872452758027(REAL64)] :
    }}}
    VAL REAL64 scale IS 10000.0(REAL64) :
    -- x is passed in microseconds, scale converts to mili and
    -- accounts for 10 times scaling
    VAL REAL64 x64 IS ((REAL64 ROUND x) * scale):
    REAL64 numer, denom, x.power :
    VALOF
      SEQ
        x.power := x64
        numer := a.coeff[0]
        denom := b.coeff[0]
        SEQ i = 1 FOR order
```

```
      SEQ
        numer := numer + (a.coeff[i] * x.power)
        denom := denom + (b.coeff[i] * x.power)
        x.power := x.power * x64
    RESULT (REAL32 ROUND (numer / denom))
:
}}}F
{{{F nag.right1.pulse
{{{  Header
-- Calculate value of first part of Right side of pulse.
-- Internal to build.mr.pulse.
}}}
REAL32 FUNCTION nag.right1.pulse (VAL REAL32 x)
  VAL INT order IS 4 : -- Order of Equation
  {{{  VAL a.coef's
  VAL []REAL64 a.coeff IS [ 1.0(REAL64),
                    -1.62558035337(REAL64),
                     0.875259954911(REAL64),
                    -0.267186288244(REAL64),
                     3.260220221397E-02(REAL64) ] :
  }}}
  {{{  VAL b.coef's
  VAL []REAL64 b.coeff IS [ 1.0(REAL64),
                    -1.58657485948(REAL64),
                     1.05682774379(REAL64),
                    -0.202667007782(REAL64),
                    -0.237394845510(REAL64) ] :
  }}}
  VAL REAL64 scale IS 10000.0(REAL64) :
  -- x is passed in microseconds, scale converts to mili and
  -- accounts for 10 times scaling
  VAL REAL64 x64 IS ((REAL64 ROUND x) * scale):
  REAL64 numer, denom, x.power :
  VALOF
    SEQ
      x.power := x64
      numer := a.coeff[0]
      denom := b.coeff[0]
      SEQ i = 1 FOR order
        SEQ
          numer := numer + (a.coeff[i] * x.power)
          denom := denom + (b.coeff[i] * x.power)
          x.power := x.power * x64
    RESULT (REAL32 ROUND (numer / denom))
:
}}}F
{{{F nag.right2.pulse
{{{  Header
-- Calculate value of second part of Right side of pulse.
-- Internal to build.mr.pulse.
}}}
REAL32 FUNCTION nag.right2.pulse (VAL REAL32 x)
  VAL INT order IS 4 : -- Order of Equation
  {{{  VAL a.coef's
  VAL []REAL64 a.coeff IS [ 1.0(REAL64),
                    -1.76363990836(REAL64),
                     0.746573193045(REAL64),
                    -0.207425786074(REAL64),
                     2.749056807106E-02(REAL64) ] :
  }}}
  {{{  VAL b.coef's
  VAL []REAL64 b.coeff IS [ 1.0(REAL64),
                    -1.71609518275(REAL64),
                     0.952224069788(REAL64),
                    -0.216933245992(REAL64),
                    -0.413199507672(REAL64) ] :
  }}}
  VAL REAL64 scale IS 10000.0(REAL64) :
  -- x is passed in microseconds, scale converts to mili and
  -- accounts for 10 times scaling
  VAL REAL64 x64 IS ((REAL64 ROUND x) * scale):
  REAL64 numer, denom, x.power :
```

```
  VALOF
   SEQ
     x.power := x64
     numer := a.coeff[0]
     denom := b.coeff[0]
     SEQ i = 1 FOR order
       SEQ
         numer := numer + (a.coeff[i] * x.power)
         denom := denom + (b.coeff[i] * x.power)
         x.power := x.power * x64
     RESULT [REAL32 ROUND (numer / denom)]
:
}}}F
}}}
{{{ decl's
VAL max.neg IS 466.0E-06(REAL32) : -- This, and next, constant specify the
VAL max.pos IS 424.0E-06(REAL32) : -- range of of the pulse
VAL disjoint IS 100.0E-06(REAL32) : -- specifies where two parts of RHS join
VAL INT series.len IS (SIZE data) :
VAL INT pulse.len IS (series.len / 1) :
VAL REAL32 head.amp.gain IS 357.0(REAL32) : -- For scaling
VAL REAL32 max.ampl IS (0.625(REAL32) / head.amp.gain) : -- For scaling
VAL REAL32 range IS (max.pos + max.neg) :
VAL REAL32 sample.width IS 2.0E-06(REAL32) :
VAL REAL32 scale IS 0.9507(REAL32) :-- ensures filtered pulse correct height
}}}
SEQ
  {{{ build negative half
  SEQ i = 0 FOR ((INT ROUND (max.neg / sample.width)) PLUS 1)
    VAL REAL32 x IS (((REAL32 ROUND i) * sample.width) - max.neg) :
    data[i] := ((neg.left.pulse(x) / scale) * max.ampl)
  }}}
  {{{ build right half
  SEQ i = (INT ROUND (max.neg / sample.width)) FOR
                        ((INT ROUND (disjoint / sample.width)) PLUS 1)
    VAL REAL32 x IS (((REAL32 ROUND i) * sample.width) - max.neg):
    data[i] := ((neg.right1.pulse(x) / scale) * max.ampl)
  SEQ i = (INT ROUND ((max.neg + disjoint) / sample.width)) FOR
        ((INT ROUND ((max.pos - disjoint) / sample.width)) PLUS 1)
    VAL REAL32 x IS (((REAL32 ROUND i) * sample.width) - max.neg):
    data[i] := ((neg.right2.pulse(x) / scale) * max.ampl)
  }}}
  {{{ zero rest of data array
  SEQ i = (INT ROUND (range / sample.width)) FOR
          ((SIZE data) - (INT ROUND (range / sample.width)))
    data[i] := 0.0(REAL32)
  }}}
:
}}}F
}}}
{{{ decl's
#USE pseudort :
#USE t4math :
{{{ VAL's
VAL INT north IS 1 :
VAL INT south IS 0 :
VAL INT mask IS #07FFFFF : -- Gives probability of 1 in 2^23, 1 in 8E+6
                  -- Used to calculate when to introduce Drop-Out
VAL REAL32 half IS 0.5(REAL32) :
VAL REAL32 T IS (one / fs) :
VAL REAL32 range IS (max.neg + max.pos) :
VAL REAL32 stan.dev IS 0.0290(REAL32) : -- Stan Dev of Amplitude Fluctuations
VAL REAL32 mean IS one :
}}}
[3 TIMES (INT ROUND (range / T))]REAL32 out.array : -- should be big enough for all instances
INT32 ran.seed :
INT time, dummy, char :
TIMER clock :
REAL32 atten, step.size, new.val :
INT data, in.ptr.to.end, out.ptr.to.end, old.dir :
INT out.ptr, in.ptr : -- Differnce between IN and OUT is amount of Data Skew.
BOOL running :
```

```
}}}
SEQ
  {{{  init
  {{{  check out array is big enough
  IF
    (SIZE out.array) < (2 TIMES (SIZE pulse))
      STOP -- for debug
    TRUE
      SKIP
  }}}
  build.ind.pulse (pulse)
  {{{  zero out.array
  SEQ i = 0 FOR (SIZE out.array)
    out.array[i] := zero
  }}}
  {{{  initialise random number stuff
  clock ? dummy
  ran.seed := (INT32 dummy)
  }}}
  out.ptr := 0
  in.ptr := (out.ptr + skew)
  old.dir := north
  running := TRUE
  }}}
  WHILE running
    SEQ
      data.in ? data -- get digital Datum
      IF
        (data = 1) OR (data = 0)
          {{{F process data
          INT trans.dir IS data :
          SEQ
            in.ptr.to.end := ((SIZE out.array) MINUS in.ptr) -- How far to end of array
            out.ptr.to.end := ((SIZE out.array) MINUS out.ptr)  -- as above
            {{{  add pulse if transition
            IF
              trans.dir <> old.dir
                {{{  transition, add pulse
                SEQ
                  {{{  calc amount of attenuation
                  SEQ
                    gen.gauss (ran.seed, stan.dev, mean, atten)
                    clock ? time
                    {{{  drop-out code
                    IF
                      (time /\ mask) = ((INT ran.seed) /\ mask)
                        atten := 0.1(REAL32) -- drop-out, -20dB
                      TRUE
                        SKIP
                    }}}
                  }}}
                  IF
                    trans.dir = north
                      {{{  add positive pulse
                      -- in two section: takes into acount problems in adding array to
                      -- non-aligned circular buffer. Amplitude fluctuation applied here.
                      IF
                        in.ptr.to.end >= (SIZE pulse)
                          SEQ i = in.ptr FOR (SIZE pulse)
                            out.array[i] := out.array[i] + (atten * pulse[i MINUS in.ptr])
                        TRUE
                          SEQ
                            SEQ i = in.ptr FOR in.ptr.to.end
                              out.array[i] := out.array[i] + (atten * pulse[i MINUS in.ptr])
                            SEQ i = 0 FOR ((SIZE pulse) MINUS in.ptr.to.end)
                              out.array[i] := out.array[i] + (atten * pulse[(i PLUS in.ptr.to.end)])
                      }}}
                    TRUE
                      {{{  add negative pulse
                      -- in two section: takes into acount problems in adding array to
                      -- non-aligned circular buffer. Amplitude fluctuation applied here (atten).
                      IF
```

```
                    in.ptr.to.end >= (SIZE pulse)
                      SEQ i = in.ptr FOR (SIZE pulse)
                        out.array[i] := out.array[i] - (atten * pulse[i MINUS in.ptr])
                    TRUE
                      SEQ
                        SEQ i = in.ptr FOR in.ptr.to.end
                          out.array[i] := out.array[i] - (atten * pulse[i MINUS in.ptr])
                        SEQ i = 0 FOR ((SIZE pulse) MINUS in.ptr.to.end)
                          out.array[i] := out.array[i] - (atten * pulse[(i PLUS in.ptr.to.end)])
                  }}}
                  old.dir := trans.dir
                }}}
              TRUE
                {{{ no transition, add nothing
                SKIP
                }}}
            }}}
            {{{ output part of out.array
            -- Ouputs section of array complete from super-position.
            -- In two part to take into account circular buffer.
            -- After data output, zero section of array.
            IF
              out.ptr.to.end >= pulse.sep
                SEQ i = out.ptr FOR pulse.sep
                  SEQ
                    data.out ! float ; out.array[i]
                    out.array[i] := zero
              TRUE
                SEQ
                  SEQ i = out.ptr FOR out.ptr.to.end
                    SEQ
                      data.out ! float ; out.array[i]
                      out.array[i] := zero
                  SEQ i = 0 FOR (pulse.sep MINUS out.ptr.to.end)
                    SEQ
                      data.out ! float ; out.array[i]
                      out.array[i] := zero
            }}}
            {{{ adjust pointers -- implements circular buffer.
            in.ptr := (in.ptr PLUS pulse.sep) REM (SIZE out.array)
            out.ptr := (out.ptr PLUS pulse.sep) REM (SIZE out.array)
            }}}
          }}}F
        data = terminate
          running := FALSE
        TRUE
          {{{ pass it on
          data.out ! int ; data
          }}}
    {{{ pass on 'terminate'
    data.out ! int ; terminate -- last instruction in process.
    }}}
:
}}}F
}}}
{{{ SC displace
{{{F displace
{{{ Header
```

-- Lateral Head Displacement Simulation.

```
-- Mixes signal according to Displacement, Write, Read tracks widths and
-- Side Field contribution.
}}}
#USE pseudort :
PROC displace ([num.tracks]CHAN OF INT.OR.FLOAT data.in,
              VAL REAL32 write, read, s.w.w,
              VAL REAL32 disp,
              [num.tracks]CHAN OF INT.OR.FLOAT data.out,
              CHAN OF ANY to.screen)
  {{{ decl's
  #USE t4math :
```

```
#USE userio :
{{{  VAL's for Track dimensions, MEASURED dimensions
VAL REAL32 safety IS ((write - read) / two) :
VAL []REAL32 track.sep IS [0.828(REAL32), 1.235(REAL32),
                           0.828(REAL32), 5.000(REAL32)] :
-- track.sep[3] for accuracy should be infinity
VAL []REAL32 guard.band IS [(track.sep[0] - write),
                            (track.sep[1] - write),
                            (track.sep[2] - write),
                            (track.sep[3] - write)] :
}}}
[num.tracks PLUS 1]REAL32 data :
[num.tracks]REAL32 a, b :
[num.tracks]INT char :
INT count, num.terminated :
BOOL running :
}}}
SEQ
  {{{  init
  {{{  calc proportions of each track
  SEQ track = 0 FOR num.tracks
    SEQ
      {{{  calc a[track]
      IF
        disp < safety
          a[track] := read
        disp < (safety + read)
          a[track] := ((safety + read) - disp)
        TRUE
          a[track] := zero
      }}}
      {{{  calc b[track]
      VAL REAL32 start IS (safety + guard.band[track]) :
      IF
        disp < start
          b[track] := zero
        disp < (start + read)
          b[track] := (disp - start)
        disp < (start + write)
          b[track] := read
        disp < ((start + read) + write)
          b[track] := (((start + read) + write) - disp)
        TRUE
          b[track] := zero
      }}}
      IF
        s.w.w > zero
          SEQ
            {{{
            {{{  calc a.side
            IF
              disp < safety
                a.side := zero
              disp < (safety + s.w.w)
                VAL REAL32 olap IS (disp - safety) :
                a.side := (olap - ((olap * olap) / (two * s.w.w)))
              disp < (safety + read)
                a.side := (s.w.w / two)
              disp < ((safety + read) + s.w.w)
                VAL REAL32 olap IS (((safety + s.w.w) + read) - disp) :
                a.side := ((olap * olap) / (two * s.w.w))
              TRUE
                a.side := zero

            }}}
            {{{  calc b.side1
            VAL REAL32 start IS (safety + guard.band[track]) :
            IF
              disp < (start - s.w.w)
                b.side1 := zero
              disp < start
                VAL REAL32 olap IS ((disp + s.w.w) - start) :
```

```
            b.side1 := ((olap * olap) / (two * s.w.w))
          disp < ((start + read) - s.w.w)
            b.side1 := (s.w.w / two)
          disp < (start + read)
            VAL REAL32 olap IS ((start + read) - disp) :
            b.side1 := (olap - ((olap * olap) / (two * s.w.w)))
          TRUE
            b.side1 := zero
        }}}
        {{{  calc b.side2
        VAL REAL32 start IS ((safety + guard.band[track]) + write) :
        IF
          disp < start
            b.side2 := zero
          disp < (start + s.w.w)
            VAL REAL32 olap IS (disp - start) :
            b.side2 := (olap - ((olap * olap) / (two * s.w.w)))
          disp < (start + read)
            b.side2 := (s.w.w / two)
          disp < ((start + read) + s.w.w)
            VAL REAL32 olap IS (((start + read) + s.w.w) - disp) :
            b.side2 := ((olap * olap) / (two * s.w.w))
          TRUE
            b.side2 := zero
        }}}
        {{{  collect, normalise, and put into array for later filing
        a[track] := (a.main + a.side)
        b[track] := ((b.main + b.side1) + b.side2)
        }}}
        }}}
      TRUE
        SKIP
    a[track] := (a[track] / read)
    b[track] := (b[track] / read)
}}}
SEQ track = 0 FOR num.tracks
  SEQ
    char[track] := 0
    sum[track] := zero
    sum.sqrs[track] := zero
    min[track] := zero
    max[track] := zero
count := 0
data[num.tracks] := zero -- ie data[4]
num.terminated := 0
running := TRUE
}}}
WHILE running
  SEQ
    {{{  get data, need to get all data from all tracks in one go.
    PAR track = 0 FOR num.tracks
      data.in[track] ? CASE
        int ; char[track]
          SKIP
        float ; data[track]
          SKIP

    }}}
    {{{  check for 'terminated'
    SEQ track = 0 FOR num.tracks
      IF
        char[track] = 0 -- initial value
          SKIP
        char[track] = terminate
          num.terminated := (num.terminated PLUS 1)
        TRUE
          SEQ
            data.out[track] ! int ; char[track]
            char[track] := 0 -- back to initial value again
    }}}
    IF
      num.terminated = 0
```

```
        {{{  process data
        SEQ
          {{{  mix signals
          SEQ track = 0 FOR num.tracks
            SEQ
              data[track] := ((data[track] * a[track]) + (data[track PLUS 1] * b[track]))
          }}}
          {{{  output signals
          PAR track = 0 FOR num.tracks
            data.out[track] ! float ; data[track]
          }}}
        }}}
      TRUE
        {{{  sink rest of data until all terminated
        -- absorbs race-conditions.
        SEQ
          WHILE num.terminated <> num.tracks
            ALT track = 0 FOR num.tracks
              data.in[track] ? CASE
                int ; char[track]
                  {{{
                  IF
                    char[track] = terminate
                      num.terminated := (num.terminated PLUS 1)
                    TRUE
                      SKIP
                  }}}
                float ; data[track]
                  SKIP
            running := FALSE
        }}}
    {{{  pass on 'terminate'
    PAR track = 0 FOR num.tracks
      data.out[track] ! int ; terminate
    }}}
:
}}}F
}}}
{{{  SC headamp
{{{F headamp
{{{  Header
```

- Head Amplifier Model.

```
-- Digital Filter. Snapshots store waveforms (after superpostion has
-- settled down), with decimation factor snapshot.step.size
}}}
PROC headamp (CHAN OF INT.OR.FLOAT data.in,
              []REAL32 pre.snapshot, post.snapshot,
              VAL INT settle.time, snapshot.step.size,
              [2]CHAN OF INT.OR.FLOAT data.out)
  {{{  decl's
  {{{  #USE's
  #USE pseudort :
  #USE t4math :
  }}}
  VAL end IS ((SIZE pre.snapshot) MINUS 1) :
  VAL REAL32 analog.mult IS 10000.0(REAL32) :
  VAL REAL32 T IS (one / fs) :
  VAL REAL32 gain IS 357.0(REAL32) : -- Gain of Headamp
  {{{  cut-off freq, un-warped, pre-warped
  VAL REAL32 fcdl IS 99.47(REAL32) :    -- in Hz
  VAL REAL32 fcdu IS 15.92E+3(REAL32) :    -- in Hz
  VAL REAL32 wcdl IS ((two * pi) * fcdl) :  -- in Rad/s, un-warped
  VAL REAL32 wcdu IS ((two * pi) * fcdu) :  -- in Rad/s, un-warped
  VAL REAL32 wcal IS ((two / T) * TAN((wcdl * T) / two)) : -- pre-warped
  VAL REAL32 wcau IS ((two / T) * TAN((wcdu * T) / two)) : -- pre-warped
  }}}
  {{{  filter coefficients
  VAL REAL32 a IS (wcal * wcau) :
  VAL REAL32 b IS (wcau - wcal) :
  VAL REAL32 c IS (two / T) :
```

```
VAL REAL32 d IS (a + (c * (b + c))) :
VAL REAL32 e IS ((b * c) / d) :
VAL REAL32 f IS ((two * (a - (c * c))) / d) :
VAL REAL32 g IS ((a + (c * (c - b))) / d) :
}}}
REAL32 xn, xn.minus.1, xn.minus.2, yn, yn.minus.1, yn.minus.2 :
INT count, ptr, char, snap.count :
BOOL settling, capturing, running :
}}}
SEQ
 {{{ init
 ptr, count, snap.count := 0, 0, 0
 xn.minus.1, yn.minus.1 := zero, zero
 xn.minus.2, yn.minus.2 := zero, zero
 settling := TRUE
 capturing := FALSE
 running := TRUE
 }}}
 WHILE running
  data.in ? CASE
   int ; char
    {{{ process char
    IF
     char = terminate
      running := FALSE
     TRUE
      {{{ pass it on
      PAR
       data.out[0] ! int ; char
       data.out[1] ! int ; char
      }}}
   }}}
   float ; xn
    {{{
    SEQ
     yn := (((e * (xn - xn.minus.2)) - (f * yn.minus.1)) - (g * yn.minus.2))
     {{{ output two copies (for the two data streams of GXO)
     VAL output IS (yn * gain) :
     SEQ
      PAR
       data.out[0] ! float ; output
       data.out[1] ! float ; output
     }}}
     {{{ snap section of the number stream
     IF
      settling
       {{{
       SEQ
        count := count PLUS 1
        IF
         settle.time > count
          SKIP
         TRUE
          SEQ
           settling := FALSE
           capturing := TRUE
       }}}
      capturing
       {{{
       SEQ
        snap.count := (snap.count PLUS 1)
        IF
         snap.count = snapshot.step.size
          SEQ
           pre.snapshot[ptr] := xn
           post.snapshot[ptr] := yn
           {{{ increment ptr
           IF
            ptr <> end
             ptr := ptr PLUS 1
            TRUE
             capturing := FALSE
```

```
                }}}
                snap.count := 0
              TRUE
                SKIP
            }}}
          TRUE
            SKIP
        }}}
        {{{  update xn.minus, yn.minus stuff
        xn.minus.2 := xn.minus.1
        xn.minus.1 := xn
        yn.minus.2 := yn.minus.1
        yn.minus.1 := yn
        }}}
      }}}
  {{{  pass on 'terminate' (two copies for two data streams if GXO)
  PAR
    data.out[0] ! int ; terminate
    data.out[1] ! int ; terminate
  }}}
:
}}}F
}}}
{{{  SC gated.cross
{{{F gated.cross
{{{  Header
```

- Gated Cross Over Detector.

```
}}}
{{{  PROC gated.cross (Parameter List)
PROC gated.cross ([2]CHAN OF INT.OR.FLOAT data.in,
                  []REAL32 gate.ana, pol.ana,
                  []BYTE gate.dig, pol.dig, gxo.dig,
                  VAL REAL32 gate.stan.dev, pol.stan.dev,
                  VAL REAL32 gate.threshold, pol.threshold,
                  VAL INT settle.time, snapshot.step.size,
                  CHAN OF INT data.out)
}}}
  {{{  CHAN decl's
  CHAN OF INT.OR.FLOAT from.gate.bandpass, from.pol.bandpass :
  CHAN OF INT from.gate, from.pol :
  }}}
  {{{  PROC decl's
  {{{  SC bandpass.gate
  {{{F bandpass.gate
  {{{  Header
```

- Band-pass filter as Differentiator.

```
  -- Filter coeff's calc at compile time.
  -- Internal to GXO
  }}}
  PROC bandpass.gate (CHAN OF INT.OR.FLOAT data.in,
                      []REAL32 snapshot,
                      VAL INT settle.time, snapshot.step.size,
                      CHAN OF INT.OR.FLOAT data.out)
    {{{  decl's
    #USE pseudort :
    #USE t4math :
    VAL end IS ((SIZE snapshot) MINUS 1) :
    VAL REAL32 analog.mult IS 10000.0(REAL32) :
    VAL REAL32 T IS (one / fs) :
    VAL REAL32 gain IS 14.2(REAL32) : -- actual gain of circuit
    {{{  frequencies, un-warped, pre-warped
    VAL REAL32 fcdl IS 10.0E+3(REAL32) :   -- in Hz
    VAL REAL32 fcdu IS 50.0E+3(REAL32) :   -- in Hz
    VAL REAL32 wcdl IS ((two * pi) * fcdl) : -- in Rad/s, un-warped
    VAL REAL32 wcdu IS ((two * pi) * fcdu) : -- in Rad/s, un-warped
    VAL REAL32 wcal IS ((two / T) * TAN((wcdl * T) / two)) : -- pre-warped
    VAL REAL32 wcau IS ((two / T) * TAN((wcdu * T) / two)) : -- pre-warped
    }}}
```

```
{{{  filter coefficients
VAL REAL32 a IS (wcal * wcau) :
VAL REAL32 b IS (wcau - wcal) :
VAL REAL32 c IS (two / T):
VAL REAL32 d IS (a + (c * (b + c))) :
VAL REAL32 e IS ((b * c) / d) :
VAL REAL32 f IS ((two * (a - (c * c))) / d) :
VAL REAL32 g IS ((a + (c * (c - b))) / d) :
}}}
REAL32 xn, xn.minus.1, xn.minus.2, yn, yn.minus.1, yn.minus.2, output :
INT count, ptr, char, snap.count :
BOOL settling, capturing, running :

}}}
SEQ
  {{{  init
  ptr, count, snap.count := 0, 0, 0
  xn.minus.1, yn.minus.1 := zero, zero
  xn.minus.2, yn.minus.2 := zero, zero
  settling := TRUE
  capturing := FALSE
  running := TRUE

  }}}
  WHILE running
    data.in ? CASE
      int ; char
        {{{  process char
        IF
          char = terminate
            running := FALSE
          TRUE
            data.out ! int ; char -- pass it on
        }}}
      float ; xn
        {{{
        SEQ
          yn := (((e * (xn - xn.minus.2)) - (f * yn.minus.1)) - (g * yn.minus.2))
          output := (yn * gain)
          data.out ! float ; output
          {{{  snap section of the number stream
          IF
            settling
              {{{
              SEQ
                count := count PLUS 1
                IF
                  settle.time > count
                    SKIP
                  TRUE
                    SEQ
                      settling := FALSE
                      capturing := TRUE
              }}}
            capturing
              {{{
              SEQ
                snap.count := (snap.count PLUS 1)
                IF
                  snap.count = snapshot.step.size
                    SEQ
                      snapshot[ptr] := output
                      {{{  increment ptr
                      IF
                        ptr <> end
                          ptr := ptr PLUS 1
                        TRUE
                          capturing := FALSE
                      }}}
                      snap.count := 0
                  TRUE
                    SKIP
```

```
          }}}
        TRUE
          SKIP
      }}}
      {{{  update history
      xn.minus.2 := xn.minus.1
      xn.minus.1 := xn
      yn.minus.2 := yn.minus.1
      yn.minus.1 := yn
      }}}
    }}}
  {{{  pass on 'terminate'
  data.out ! int ; terminate
  }}}
:
}}}F
}}}
{{{  SC bandpass.pol
{{{F bandpass.pol
{{{  Header
```

– Band-pass filter as gain block.

```
-- Filter coeff's calc at compile time.
-- Internal to GXO
}}}
PROC bandpass.pol (CHAN OF INT.OR.FLOAT data.in, []REAL32 snapshot,
            VAL INT settle.time, snapshot.step.size,
            CHAN OF INT.OR.FLOAT data.out)
 {{{  decl's
 #USE pseudort :
 #USE t4math :
 VAL end IS ((SIZE snapshot) MINUS 1) :
 VAL REAL32 analog.mult IS 10000.0(REAL32) :
 VAL REAL32 T IS (one / fs) :
 VAL REAL32 gain IS 3.6(REAL32) : -- actual gain of circuit 3/10/90
 {{{  frequencies, un-warped, pre-warped
 VAL REAL32 fcdl IS 102.6(REAL32) :    -- in Hz
 VAL REAL32 fcdu IS 6.792E+3(REAL32) :    -- in Hz
 VAL REAL32 wcdl IS ((two * pi) * fcdl) :  -- in Rad/s, un-warped
 VAL REAL32 wcdu IS ((two * pi) * fcdu) :  -- in Rad/s, un-warped
 VAL REAL32 wcal IS ((two / T) * TAN((wcdl * T) / two)) : -- pre-warped
 VAL REAL32 wcau IS ((two / T) * TAN((wcdu * T) / two)) : -- pre-warped
 }}}
 {{{  filter coefficients
 VAL REAL32 a IS (wcal * wcau) :
 VAL REAL32 b IS (wcau - wcal) :
 VAL REAL32 c IS (two / T) :
 VAL REAL32 d IS (a + (c * (b + c))) :
 VAL REAL32 e IS ((b * c) / d) :
 VAL REAL32 f IS ((two * (a - (c * c))) / d) :
 VAL REAL32 g IS ((a + (c * (c - b))) / d) :
 }}}
 REAL32 xn, xn.minus.1, xn.minus.2, yn, yn.minus.1, yn.minus.2, output :
 INT count, ptr, char, snap.count :
 BOOL settling, capturing, running :
 }}}
 SEQ
  {{{  init
  ptr, count, snap.count := 0, 0, 0
  xn.minus.1, yn.minus.1 := zero, zero
  xn.minus.2, yn.minus.2 := zero, zero
  settling := TRUE
  capturing := FALSE
  running := TRUE
  }}}
  WHILE running
   data.in ? CASE
    int ; char
     {{{  process char
     IF
      char = terminate
```

```
          running := FALSE
        TRUE
          data.out ! int ; char
      }}}
    float ; xn
      {{{
      SEQ
        yn := (((e * (xn - xn.minus.2)) - (f * yn.minus.1)) - (g * yn.minus.2))
        output := (yn * gain)
        data.out ! float ; output
        {{{  snap section of the number stream
        IF
          settling
            {{{
            SEQ
              count := count PLUS 1
              IF
                settle.time > count
                  SKIP
                TRUE
                  SEQ
                    settling := FALSE
                    capturing := TRUE
            }}}
          capturing
            {{{
            SEQ
              snap.count := (snap.count PLUS 1)
              IF
                snap.count = snapshot.step.size
                  SEQ
                    snapshot[ptr] := output
                    {{{  increment ptr
                    IF
                      ptr <> end
                        ptr := ptr PLUS 1
                      TRUE
                        capturing := FALSE
                    }}}
                    snap.count := 0
                TRUE
                  SKIP
            }}}
          TRUE
            SKIP
        }}}
        {{{  update history
        xn.minus.2 := xn.minus.1
        xn.minus.1 := xn
        yn.minus.2 := yn.minus.1
        yn.minus.1 := yn
        }}}
      }}}
  {{{  pass on 'terminate'
  data.out ! int ; terminate
  }}}
:
}}}F
}}}
{{{  SC comp
{{{F comp
{{{  Header
```

-- Comparator model with Hysteresis.

```
-- Thresholding value input at run time.
-- Internal to GXO.
-- Stores snapshot of waveforms after 'settle.time's samples
-- has passed, decimating by snapshot.step.size
}}}
PROC comp (CHAN OF INT.OR.FLOAT data.in,
        VAL REAL32 stan.dev, threshold,
```

```
      VAL INT settle.time, snapshot.step.size,
      []BYTE snapshot,
      CHAN OF INT data.out)
{{{  SC gen.gauss
{{{F gen.gauss
{{{  Header
```

– Generate Gaussian noise (with zero mean).

```
-- Internal to Comparator.
}}}
PROC gen.gauss (INT32 seed,
            VAL REAL32 stan.dev,
            REAL32 norm.num)
  #USE t4math :
  #USE pseudort :
  VAL INT k IS 12 : --number of random numbers summed to produce Normal Dist
  VAL REAL32 twelve IS 12.0(REAL32) :
  VAL REAL32 shift IS ((REAL32 ROUND k) / two) :
  -- VAL REAL32 divisor IS SQRT ((REAL32 ROUND k) / twelve) : -- when k <> 12
  SEQ
    norm.num := 0.0(REAL32)
    SEQ j = 0 FOR k
      REAL32 temp :
      SEQ
        temp, seed := RAN (seed)
        norm.num := norm.num + temp
    norm.num := ((norm.num - shift) * stan.dev)
    -- norm.num := (((norm.num - shift) / divisor) * stan.dev)
:
}}}F
}}}
{{{  decl's
#USE pseudort :
VAL REAL32 pos.thresh IS (zero + threshold) :
VAL REAL32 neg.thresh IS (zero - threshold) :
VAL INT pos IS 1 :
VAL INT neg IS 0 :
VAL end IS ((SIZE snapshot) MINUS 1) :
VAL REAL32 mean IS 0.0(REAL32) :
INT32 ran.seed :
REAL32 noise, number :
INT dummy, count, ptr, char, output, snap.count :
BOOL settling, capturing, running :
TIMER clock :
}}}
SEQ
  {{{  init
  {{{  initialise random number stuff
  clock ? dummy
  ran.seed := (INT32 dummy)
  }}}
  ptr, count, snap.count := 0, 0, 0
  output := pos
  settling := TRUE
  capturing := FALSE
  running := TRUE
  }}}
  WHILE running
    data.in ? CASE
      int ; char
        {{{
        IF
          char = terminate
            running := FALSE
          TRUE
            data.out ! char -- pass it on
        }}}
      float ; number
        {{{
        SEQ
          {{{  add noise
```

```
            gen.gauss (ran.seed, stan.dev, noise)
            number := (number + noise)
            }}}
            {{{  calc new output
            IF
              output = pos
                IF
                  neg.thresh > number
                    output := neg
                  TRUE
                    SKIP
              TRUE
                IF
                  number > pos.thresh
                    output := pos
                  TRUE
                    SKIP
            }}}
            data.out ! output
            {{{  copy section of number stream
            IF
              settling
                {{{
                SEQ
                  count := count PLUS 1
                  IF
                    settle.time > count
                      SKIP
                    TRUE
                      SEQ
                        settling := FALSE
                        capturing := TRUE
                }}}
              capturing
                {{{
                SEQ
                  snap.count := (snap.count PLUS 1)
                  IF
                    snap.count = snapshot.step.size
                      SEQ
                        snapshot[ptr] := (BYTE output)
                        {{{  increment ptr
                        IF
                          ptr <> end
                            ptr := ptr PLUS 1
                          TRUE
                            capturing := FALSE
                        }}}
                        snap.count := 0
                    TRUE
                      SKIP
                }}}
              TRUE
                SKIP
            }}}
          }}}
        {{{  pass on 'terminate'
        data.out ! terminate
        }}}
:
}}}F
}}}
{{{  SC gate.out
{{{F gate.out
{{{  Header
```

– D-Type Flip-Flop Model.

```
-- Internal to GXO.
-- Inputs Gating signal and Polarity signal, outputs polarity
-- when gating changes.
}}}
```

```
PROC gate.out (CHAN OF INT from.gate, from.pol,
               []BYTE snapshot,
               VAL INT settle.time, snapshot.step.size,
               CHAN OF INT data.out)
  {{{  decl's
  #USE pseudort :
  VAL end IS ((SIZE snapshot) MINUS 1) :
  INT count, ptr, snap.count :
  INT polarity, gate, last.gate, output :
  BOOL settling, capturing, running :
  }}}
  SEQ
    {{{  initialise
    ptr, count, snap.count := 0, 0, 0
    output , last.gate := 0, 0
    settling := TRUE
    capturing := FALSE
    running := TRUE
    }}}
    WHILE running
      SEQ
        {{{  get both parts of data
        PAR
          from.pol ? polarity
          from.gate ? gate
        }}}
        IF
          (gate = 0) OR (gate = 1)
            {{{  process
            SEQ
              IF
                gate = last.gate
                  SKIP
                TRUE
                  SEQ
                    output := polarity
                    last.gate := gate
              data.out ! output
              {{{  snap section of the number stream
              IF
                settling
                  {{{
                  SEQ
                    count := count PLUS 1
                    IF
                      settle.time > count
                        SKIP
                      TRUE
                        SEQ
                          settling := FALSE
                          capturing := TRUE
                  }}}
                capturing
                  {{{
                  SEQ
                    snap.count := (snap.count PLUS 1)
                    IF
                      snap.count = snapshot.step.size
                        SEQ
                          snapshot[ptr] := (BYTE output)
                          {{{  increment ptr
                          IF
                            ptr <> end
                              ptr := ptr PLUS 1
                            TRUE
                              capturing := FALSE
                          }}}
                          snap.count := 0
                      TRUE
                        SKIP
                  }}}
                TRUE
```

```
            SKIP
          }}}
        }}}
      gate = terminate
        running := FALSE
      TRUE
        {{{ pass it on
        data.out ! gate
        }}}
  {{{ pass on terminate
  data.out ! terminate
  }}}
:
}}}F
}}}
```

-- The following PARallel forms the

-- Gated Cross-Over Detector Model.

```
}}}
PAR
  {{{ bandpass d/dt              v
  bandpass.gate (data.in[0], gate.ana, settle.time,
              snapshot.step.size, from.gate.bandpass)
  }}}
  {{{      v              bandpass filt
  bandpass.pol (data.in[1], pol.ana, settle.time,
              snapshot.step.size, from.pol.bandpass)
  }}}
  {{{   comparator              v
  comp (from.gate.bandpass, gate.stan.dev, gate.threshold, settle.time,
            snapshot.step.size, gate.dig, from.gate)
  }}}
  {{{      v              comparator
  comp (from.pol.bandpass, pol.stan.dev, pol.threshold,
          settle.time, snapshot.step.size, pol.dig, from.pol)
  }}}
  {{{      >-- gate out polarity --<
  gate.out (from.gate, from.pol, gxo.dig, settle.time,
          snapshot.step.size, data.out)
  }}}
:
}}}F
}}}
{{{ SC sampled.2.event
{{{F sampled.2.event
{{{ Header
```

-- Sampled-to-Event Interface.

```
-- Inputs data in Sampled Data stream, monitors for Events,
-- converts to event data stream, stores in Data and Time arrays.
-- Can initiate termination sequence (via stop.prbs) when arrays full.
-- All sections are robust to receiving odd numbers of samples from different
-- channels. Termination only when terminate signal received from all tracks.
}}}
PROC sampled.2.event ([]CHAN OF INT data.in,
                []BYTE data,
                []INT times,
                CHAN OF INT stop.prbs,
                CHAN OF ANY to.screen)
  {{{ decl's
  #USE pseudort :
  #USE userio :
  VAL INT end.of.array IS ((SIZE data) MINUS 1) :
  VAL REAL32 T IS (one / fs) :
  VAL REAL32 T.hp.tps IS ((REAL32 ROUND hp.tps) * T) : -- simulates High Priority timing Resolution
  VAL REAL32 range IS (max.neg + max.pos) :
  VAL INT samples.to.throw.away IS (2 TIMES (INT ROUND (range / T))) :
                                -- This is twice minimum for safety
  REAL32 time :
```

```
[num.tracks]INT char :
INT num.terminated, ptr :
INT input, last.input :
BOOL capturing, freewheeling :
}}}
SEQ
  {{{ init
  time := zero
  last.input := 0
  num.terminated := 0
  ptr := 0
  capturing := TRUE
  freewheeling := TRUE
  {{{ throw away first block of samples.
  INT count :
  BOOL throwing.away :
  SEQ
    throwing.away := TRUE
    count := 0
    WHILE throwing.away
      SEQ
        {{{ get all track data in parallel
        PAR track = 0 FOR num.tracks
          data.in[track] ? char[track]
        }}}
        {{{ check track data one by one
        input := 0
        SEQ track = 0 FOR num.tracks
          VAL char.track IS char[track] :
          IF
            (char.track = 0) OR (char.track = 1)
              SEQ
                input := (input \/ (char.track << track))
                {{{ check number of samples thrown away
                count := count PLUS 1
                IF
                  count = samples.to.throw.away
                    SEQ
                      throwing.away := FALSE
                      last.input := (input /\ #0F) -- ready for first valid
                  TRUE
                    SKIP
                }}}
            char.track = terminate
              {{{ check how many terminated
              SEQ
                throwing.away := FALSE
                capturing := FALSE
                num.terminated := (num.terminated PLUS 1)
                IF
                  num.terminated = num.tracks
                    freewheeling := FALSE
                  TRUE
                    SKIP
              }}}
            TRUE
              {{{ pass to screen IF track[0]
              IF
                track = 0
                  write.char (to.screen, (BYTE char[0]))
                TRUE
                  SKIP
              }}}
      }}}
  }}}
  }}}
  {{{ put data into array, then when full...
  WHILE capturing
    SEQ
      time := (time + T.hp.tps) -- keep time-base up to date
      {{{ build input word
      PAR track = 0 FOR num.tracks -- must get all in parallel
```

```
      data.in[track] ? char[track]
    input := 0
    SEQ track = 0 FOR num.tracks  -- combine in to one word
      VAL INT char.track IS char[track] :
      IF
        (char.track = 0) OR (char.track = 1)
          input := (input \/ (char.track << track))
        char.track = terminate
          {{{  check how many terminated
          SEQ
            capturing := FALSE
            num.terminated := (num.terminated PLUS 1)
            IF
              num.terminated = num.tracks
                freewheeling := FALSE
              TRUE
                SKIP
          }}}
        TRUE
          {{{  pass to screen IF track[0]
          IF
            track = 0
              write.char (to.screen, (BYTE char[0]))
            TRUE
              SKIP
          }}}
      input := (input /\ #0F)
    }}}
    IF
      (input <> last.input) AND capturing
        {{{  put time and date into array
        SEQ
          data[ptr] := (BYTE last.input) -- lastinput because of one bit buffer in hardware.
          last.input := input -- ready for next transition test
          times[ptr] := (INT ROUND time)
          {{{  update ptr
          IF
            ptr <> end.of.array
              ptr := ptr PLUS 1
            TRUE
              capturing := FALSE
          }}}
        }}}
      TRUE
        SKIP
  }}}
  {{{  clean termination
  IF
    freewheeling
      {{{  throw away data AND output 'terminate'
      PAR
        stop.prbs ! terminate
        {{{  throw away rest of data
        WHILE freewheeling
          ALT track = 0 FOR num.tracks
            data.in[track] ? char[track]
              IF
                char[track] <> terminate
                  SKIP -- throw it away
                TRUE
                  {{{  check how many terminated
                  SEQ
                    num.terminated := (num.terminated PLUS 1)
                    IF
                      num.terminated = num.tracks
                        freewheeling := FALSE
                      TRUE
                        SKIP
                  }}}
      }}}
    }}}
    TRUE
```

```
    SKIP
  :]]]
:
}]]F
}]]
  {{{
```

– The following PARallel construct models

-- the replay process.

```
  PAR
    manch.coded.prbs (keyboard, stop.prbs, prbs.reg.len, staggered, from.prbs)
    PAR track = 0 FOR num.tracks
      read (from.prbs[track], basic.pulse[track],
            pulse.sep, skew.samples[track], from.read[track])
    displace (from.read, write.width, read.width, side.write.width,
              displacement, from.disp, to.log)
    PAR track = 0 FOR num.tracks
      PAR
        headamp (from.disp[track],
                 [pre.headamp[track] FROM 0 FOR (SIZE pre.headamp[track])],
                  [post.headamp[track] FROM 0 FOR (SIZE post.headamp[track])],
                    settle.time, snapshot.step.size, from.headamp[track])
        gated.cross (from.headamp[track],
                     [gate.ana[track] FROM 0 FOR snapshot.len],
                      [pol.ana[track] FROM 0 FOR snapshot.len],
                       [gate.dig[track] FROM 0 FOR snapshot.len],
                        [pol.dig[track] FROM 0 FOR snapshot.len],
                         [gxo.dig[track] FROM 0 FOR snapshot.len],
                          gate.stan.dev, pol.stan.dev,
                           gate.threshold, pol.threshold,
                            settle.time, snapshot.step.size,
                             from.gxo[track])
      sampled.2.event (from.gxo, [data FROM 0 FOR block.size],
                       [times FROM 0 FOR block.size], stop.prbs, to.log)
    :]]]
  :]]]
}]]F
{{{ check data
{{{ message
print.elapsed.time (to.log, start.time)
write.text.line (to.log, "          Checking data....")
}]]
{{{ PROC decl's
{{{ SC distib
{{{F distib
{{{ Header
```

– Distribute Event Times and Data,

```
– held in arrays to relevent track.
– Initiates termintaion sequence when arrays are empty.
}]]
#USE pseudort :
PROC distrib ([]INT tarray,
              []BYTE darray,
              [num.tracks]CHAN OF INT data.out)
  {{{ Decl's
  VAL first.valid IS 9 : – Let model settle down.
  VAL start IS (first.valid PLUS 1) :
  INT old.data, track.mask :
  :]]]
  SEQ
    {{{ Initialise
    old.data := INT (darray[first.valid])
    :]]]
    SEQ sample = start FOR ((SIZE tarray) MINUS start)
      VAL new.data IS INT (darray[sample]) :
      SEQ
        track.mask := #01
        SEQ track = 0 FOR num.tracks
```

```
      SEQ
        IF -- this should be redundant, there should always be a difference
          ( (new.data /\ track.mask) <> (old.data /\ track.mask) )
            {{{  output data to relavant channel
            data.out[track] ! (tarray[sample MINUS 1] << 1) \/
                        ((new.data /\ track.mask) >> track)
            }}}
          TRUE
            SKIP
        track.mask := track.mask << 1 -- adjust bit mask for next track
      old.data := new.data
    {{{  send 'terminate' to all channels now arrays empty
    SEQ track = 0 FOR num.tracks
      data.out[track] ! terminate
    }}}
:
}}}F
}}}
{{{  SC decode
{{{F decode
{{{  Header
```

- Bi-Phase-L Channel Decoder.

```
-- Used simulated data rate to calculate sample points.
}}}
PROC man.dec (VAL INT sim.data.rate,
              CHAN OF INT data.in, data.out)
  {{{  decl's
  #USE pseudort :
  VAL data.bit.time IS (hp.tps / sim.data.rate) : -- captured at High Priority
  VAL sample.period IS data.bit.time :
  VAL code.period IS (data.bit.time >> 1) : -- ie half
  VAL margin IS (code.period >> 1) :
  VAL max.drop.out IS 100 : -- ie number code bits, used for memory allocation
  INT t1, t2, t3, p1.len, p2.len, sample.time :
  INT data, track.data :
  INT p1.units :
  BOOL running, initialising :
  }}}
  SEQ
    {{{  Initialise
    running := TRUE
    initialising := TRUE
    {{{  get t1
    data.in ? data
    IF
      data <> terminate
        t1 := (data >> 1)
      TRUE
        SEQ
          running := FALSE
          initialising := FALSE

    }}}
    WHILE initialising
      {{{  look for interval of 2
      SEQ
        data.in ? data
        IF
          data <> terminate
            {{{  look for period '2'
            SEQ
              t2 := (data >> 1)
              p1.len := (t2 MINUS t1)
              IF
                {{{  1.75 < p1.len < 2.25 -- Widow only half normal during synchronisation
                ( ( (p1.len MINUS (margin >> 1) ) < sample.period) AND
                  ( (p1.len PLUS (margin >> 1) ) > sample.period) )
                }}}
                  {{{  found period 2 long
                  SEQ
```

```
            initialising := FALSE
            sample.time := (t2 PLUS sample.period)
            p1.units := 2
           }}}
         TRUE
           t1 := t2
      }}}
    TRUE
      SEQ
        running := FALSE
        initialising := FALSE
 }}}
}}}
WHILE running
 SEQ
  data.in ? data
  IF
   data <> terminate
    {{{ process data
    SEQ
     t3 := (data >> 1) -- seperate data from time
     track.data := (data /\ #01) -- seperate time from data
     p2.len := (t3 MINUS t2)
     IF
      IF p2.units = 0 FOR max.drop.out -- calcs p2 in terms of data period
       ( (code.period TIMES p2.units) PLUS margin) >= p2.len
        SEQ
         {{{ process p2.units
         IF
          {{{ 1:2 or 2:2
          (p2.units = 2)
          }}}
           {{{
           SEQ
            data.out ! track.data
            {{{ recalc period
            SKIP
            }}}
            sample.time := (t3 PLUS sample.period)
           }}}
          {{{ 2:1
          ((p1.units = 2) AND (p2.units = 1))
          }}}
           {{{
           SEQ
            {{{ recalc period
            SKIP
            }}}
            sample.time := (t3 PLUS (sample.period >> 1) )
           }}}
          {{{ 1:1
          ((p1.units = 1) AND (p2.units = 1))
          }}}
           {{{
           SEQ
            {{{ recalc period
            SKIP
            }}}
            IF
             {{{ past sample point
             (t3 PLUS margin) > sample.time
             }}}
              {{{
              SEQ
               data.out ! track.data
               sample.time := (t3 PLUS sample.period)
              }}}
             TRUE -- else, another transition will occur
              SKIP -- before sample point
           }}}
          TRUE -- zero or 2 < p2.units < max drop out
           {{{ freewheel for corrupted data
```

```
                    WHILE (t3 PLUS margin) > sample.time
                      SEQ
                        data.out ! track.data
                        sample.time := (sample.time PLUS sample.period)
                    }}}
                  }}}
                  p1.units := p2.units
              TRUE -- p2 > max.drop.out units long
                SKIP
            t2 := t3
          }}}
        TRUE
          {{{ shutdown
          running := FALSE
          }}}
    {{{ pass on 'terminate'
    data.out ! terminate
    }}}
:
}}}F
}}}
{{{ SC prbs.check
{{{F prbs.check
{{{ Header
```

- PRBS Error Check and Classification.

```
-- Regenerates PRBS, compares with incoming data, classifies errors.
}}}
PROC prbs.check (VAL INT reg.len, max.bad, min.good,
                 CHAN OF INT data.in,
                 INT count, class.good, class.bad,
                 INT good.bits, bad.bits, lost.synch,
                 []INT burst.his)
  {{{ PROC defs
  -- init.pointers, move.pointers declaration removed.
  -- Same as in gen.prbs.
  }}}
 {{{ decl's
  #USE pseudort :
  #USE userio :
  VAL his.len IS (max.bad PLUS 1) : -- History Length
  [16]INT s.reg : -- Shift Register
  BOOL running, in.synch, not.in.error :
  INT fb1, fb2, next.fb : -- Register Pointers
  INT data, new.prbs.bit :
  INT ass.good.bits, ass.bad.bits : -- Intermediatte/ASSummed Results
  INT recover.len, burst.len :
  }}}
  SEQ
    {{{ initilise process
    class.good, class.bad := 0, 0
    good.bits, bad.bits := 0, 0
    SEQ i = 0 FOR his.len
      burst.his[i] := 0
    lost.synch, count := 0, 0
    running:= TRUE
    }}}
    WHILE running
      SEQ
        {{{ initialise this stream
        init.pointers (reg.len, fb1, fb2, next.fb)
        ass.good.bits, ass.bad.bits := 0, 0
        recover.len, burst.len := 0, 0
        {{{ fill s.reg with incoming data, whilst monitoring for Termination
        {{{ decl's
        INT index :
        BOOL filling :
        }}}
        SEQ
          {{{ init
          index := 0
```

```
filling := TRUE
}}}
WHILE filling
  SEQ
    data.in ? data
    IF
      data <> terminate
        {{{  put into f/b reg
        SEQ
          s.reg[(reg.len MINUS 1) MINUS index] := data
          {{{  increment index
          index := (index PLUS 1)
          IF
            index = reg.len
              SEQ
                filling := FALSE
                in.synch := TRUE
                not.in.error := TRUE
            TRUE
              SKIP
          }}}
        }}}
      TRUE
        {{{  end loops
        SEQ
          filling := FALSE
          running := FALSE
          in.synch := FALSE
        }}}
}}}
}}}
WHILE in.synch
  SEQ
    data.in ? data
    IF
      data <> terminate
        {{{  process data
        SEQ
          count := count PLUS 1
          {{{  clock PRBS
          SEQ
            move.pointers (fb1, fb2, next.fb)
            new.prbs.bit := s.reg[fb1) >< s.reg[fb2] -- EXCLUSIVE OR
            s.reg[next.fb] := new.prbs.bit
          }}}
          IF
            not.in.error
              {{{
              IF
                {{{  new bit OK
                (new.prbs.bit = data)
                }}}
                  {{{
                  SEQ
                    class.good := (class.good PLUS 1)
                    good.bits := (good.bits PLUS 1)
                  }}}
                TRUE
                  {{{
                  SEQ
                    not.in.error := FALSE
                    ass.good.bits, recover.len := 0, 0
                    ass.bad.bits, burst.len := 1, 1
                  }}}
              }}}
            TRUE -- ie in.error
              {{{
              SEQ
                IF
                  {{{  new data OK
                  (new.prbs.bit = data)
                  }}}
```

```
                    {{{
                    SEQ
                      recover.len := (recover.len PLUS 1)
                      IF
                        recover.len = min.good
                          {{{ burst finished, record errors
                          SEQ
                            burst.len := (burst.len MINUS (min.good MINUS 1))
                            burst.his[burst.len] := (burst.his[burst.len] PLUS 1)
                            class.good := (class.good PLUS recover.len)
                            class.bad := (class.bad PLUS burst.len)
                            good.bits := (good.bits PLUS (ass.good.bits PLUS 1))
                            bad.bits := (bad.bits PLUS ass.bad.bits)
                            not.in.error := TRUE
                          }}}
                        TRUE
                          SEQ
                            ass.good.bits := (ass.good.bits PLUS 1)
                            burst.len := (burst.len PLUS 1)
                    }}}
                  TRUE -- new data wrong
                    {{{
                    SEQ
                      burst.len := (burst.len PLUS 1)
                      IF
                        burst.len > max.bad
                          SEQ
                            in.synch := FALSE
                            lost.synch := (lost.synch PLUS 1)
                        TRUE
                          SEQ
                            ass.bad.bits := (ass.bad.bits PLUS 1)
                            recover.len := 0
                    }}}
                }}}
            }}}
          TRUE
            {{{ end loops
            SEQ
              running := FALSE
              in.synch := FALSE
            }}}
:
}}}F
}}}
}}}
CHAN OF INT from.tee :
[num.tracks]CHAN OF INT from.distrib, from.decode :
```

- The following PARallel Construct implements

  -- the Decodes and Error Checking.

```
PAR
  distrib ([times FROM 0 FOR block.size], [data FROM 0 FOR block.size],
           from.distrib)
  PAR track = 1 FOR 3
    PAR
      man.dec (sim.data.rate, from.distrib[track], from.decode[track])
      prbs.check (prbs.reg.len, max.bad, min.good,
                  from.decode[track],
                  count[track], class.good[track], class.bad[track],
                  good.bits[track], bad.bits[track],
                  lost.synch[track], burst.his[track])
  man.dec (sim.data.rate, from.distrib[0], from.decode[0])
  prbs.check (prbs.reg.len, max.bad, min.good, from.decode[0],
              count[0], class.good[0], class.bad[0],
              good.bits[0], bad.bits[0],
              lost.synch[0], burst.his[0])
}}}
{{{ calculate rates
calc.rates (class.good, class.bad, lost.synch,
```

```
          max.bad, block.size, track.rate, rate)
}}}
{{{ print totals
print.totals (count, lost.synch, class.good, class.bad, good.bits, bad.bits,
          burst.his, his.len, track.rate, rate, to.log)
}}}
{{{ file waveforms arrays
VAL []BYTE dont.file.str IS "do not file waveforms              " :
IF
  {{{ dont want waveforms filed....
  eqstr ([waveform.filename FROM 0 FOR waveform.name.len],
          [dont.file.str FROM 0 FOR waveform.name.len])
  }}}
    {{{
    write.text.line (to.log, "Waveforms NOT filed.")
    }}}
  TRUE
    {{{ file waveforms
    {{{ local decl's
    CHAN OF ANY to.waveforms :
    INT subplot.num, mesg.num, waveform.filer.result :
    REAL32 xorig, yorig :
    }}}
    {{{ PROC decl's
    {{{ SC draw.graphs
    {{{F draw.graphs
    {{{ Header
```

- Draw Graphs.

```
    -- Takes snapshot data in arrays, output in TellaGraf form.
    -- Starting at XOrigin,YOrigin. Each graph has unique SUBPLOT number.
    }}}
    PROC draw.graphs ([][]REAL32 pre.headamp, post.headamp,
                      [][]REAL32 gate.ana, pol.ana,
                      [][]BYTE pol.dig, gate.dig, gxo.dig,
                      VAL INT snapshot.len,
                      REAL32 xorig, yorig,
                      INT subplot.num,
                      CHAN OF ANY data.out)
      {{{ VAL's
      VAL INT num.tracks IS (SIZE pol.op) :
      VAL REAL32 ylen IS 2.5(REAL32) : -- y axis length
      VAL REAL32 xlen IS 18.0(REAL32) : -- x axis length
      }}}
      {{{ SC plot analogue wave
      {{{F plot analogue wave
      {{{ Header
      -- Wraps floating point numbers for TellaGraf PLOT.
      -- Internal to Draw.Graphs
      }}}
      PROC plot.a.wave (VAL []REAL32 analogue,
                        VAL REAL32 xorig, yorig, xlen, ylen,
                        INT subplot.num,
                        CHAN OF ANY data.out)

        {{{ SC write.float.array
        {{{F write.float.array
        {{{ Header
        -- Outputs 'per.line' floating point numbers per line
        -- Internal to plot.analogue
        }}}
        PROC write.float.array (VAL []REAL32 array,
                                VAL INT per.line,
                                CHAN OF ANY data.out)
          {{{ decl's
          #USE userio :
          }}}
          SEQ i = 0 FOR ((SIZE array) / per.line)
            SEQ
              SEQ j = 0 FOR per.line
                VAL INT index IS ((i * per.line) PLUS j) :
```

```
      SEQ
        write.real32 (data.out, array[index], 0, 4)
        write.full.string (data.out, " ")
    newline (data.out)
:
}}}F
}}}
#USE userio :
{{{  VAL's
VAL INT floats.per.line IS 6 :
VAL INT analog.mult IS INT 10000 :
VAL INT analog.y IS ((3 TIMES analog.mult) / 2) : -- ie plus 50%
}}}
SEQ
  {{{  continue
  write.full.string (data.out, "CONTINUE ")
  write.int (data.out, subplot.num, 0)
  write.text.line (data.out, ".")
  subplot.num := (subplot.num PLUS 1)
  }}}
  write.text.line (data.out, "GENERATE A PLOT.")
  write.text.line (data.out, "SEQUENCE DATA.")
  write.text.line (data.out, "*"ANALOGUE*"")
  write.float.array (analogue, floats.per.line, data.out)
  write.text.line (data.out, "END OF DATA.")
  {{{  x axis
  write.full.string (data.out, "X AXIS ORIGIN ")
  write.real32 (data.out, xorig, 0, 0 )
  write.full.string (data.out, ", LENGTH ")
  write.real32 (data.out, xlen, 0, 0)
  write.text.line (data.out, ", OFF.")
  }}}
  {{{  yaxis
  write.full.string (data.out, "Y AXIS ORIGIN ")
  write.real32 (data.out, yorig, 0, 0)
  write.full.string (data.out, ", LENGTH ")
  write.real32 (data.out, ylen, 0, 0)
  write.full.string (data.out, ", MIN ")
  write.int (data.out, (0 MINUS analog.y), 0)
  write.full.string (data.out, ", STEP ")
  write.int (data.out, analog.y, 0)
  write.full.string (data.out, ", MAX ")
  write.int (data.out, (0 PLUS analog.y), 0)
  write.text.line (data.out, ".")
  }}}
  {{{  subplot
  write.full.string (data.out, "SUBPLOT ")
  write.int (data.out, subplot.num, 0)
  write.text.line (data.out, ".")
  }}}
:
}}}F
}}}
{{{  SC plot digital wave
{{{F plot digital wave
{{{  Header
-- Internal to Draw.Graphs.
-- Takes Integer array, output with text for TealGraf.
}}}
PROC plot.d.wave (VAL []BYTE digital,
                  VAL REAL32 xorig, yorig, xlen, ylen,
                  INT subplot.num,
                  CHAN OF ANY data.out)

  {{{  SC write.int.array
  {{{F write.int.array
  {{{  Header
  -- Internal to plot.digital.
  -- Outputs Integer array, 'per.line' INTs per line.
  }}}
  PROC write.int.array (VAL []BYTE array,
                        VAL INT per.line,
```

```
                    CHAN OF ANY data.out)
  {{{  decl's
  #USE userio :
  }}}
  SEQ i = 0 FOR ((SIZE array) / per.line)
    SEQ
      SEQ j = 0 FOR per.line
        VAL INT index IS ((i * per.line) PLUS j) :
        SEQ
          write.int (data.out, (INT array[index]), 2)
          write.full.string (data.out, " ")
      newline (data.out)
:
}}}F
}}}
#USE userio :
VAL INT ints.per.line IS 20 :
SEQ
  {{{  subplot
  write.full.string (data.out, "CONTINUE ")
  write.int (data.out, subplot.num, 0)
  write.text.line (data.out, ".")
  subplot.num := (subplot.num PLUS 1)
  }}}
  write.text.line (data.out, "GENERATE A PLOT.")
  write.text.line (data.out, "SEQUENCE DATA.")
  write.text.line (data.out, "*"DIGITAL*"")
  write.int.array (digital, ints.per.line, data.out)
  write.text.line (data.out, "END OF DATA.")
  {{{  x axis
  write.full.string (data.out, "X AXIS ORIGIN ")
  write.real32 (data.out, xorig, 0, 0 )
  write.full.string (data.out, ", LENGTH ")
  write.real32 (data.out, xlen, 0, 0)
  write.text.line (data.out, ", ANNOTATION OFF, EXISTENCE OFF.")
  }}}
  {{{  yaxis
  write.full.string (data.out, "Y AXIS ORIGIN ")
  write.real32 (data.out, yorig, 0, 0)
  write.full.string (data.out, ", LENGTH ")
  write.real32 (data.out, ylen, 0, 0)
  write.full.string (data.out, ", MIN ")
  write.real32 (data.out, -2.0(REAL32), 0, 0)
  write.full.string (data.out, ", STEP ")
  write.real32 (data.out, 1.0(REAL32), 0, 0)
  write.full.string (data.out, ", MAX ")
  write.real32 (data.out, 2.0(REAL32), 0, 0)
  write.text.line (data.out, ", EXISTENCE OFF, OFF.")
  }}}
  {{{  subplot
  write.full.string (data.out, "SUBPLOT ")
  write.int (data.out, subplot.num, 0)
  write.text.line (data.out, ".")
  }}}
:
}}}F
}}}
SEQ
  SEQ track = 0 FOR num.tracks
    {{{  output gate signals
    SEQ
      plot.a.wave ([gate.ana[track] FROM 0 FOR snapshot.len],
                   xorig, yorig, xlen, ylen, subplot.num, data.out)
      plot.d.wave ([gate.dig[track] FROM 0 FOR snapshot.len],
                   xorig, yorig, xlen, ylen, subplot.num, data.out)
      yorig := (yorig - ylen)
    }}}
  SEQ track = 0 FOR num.tracks
    {{{  output polarity signals
    SEQ
      plot.a.wave ([pol.ana[track] FROM 0 FOR snapshot.len],
                   xorig, yorig, xlen, ylen, subplot.num, data.out)
```

```
      plot.d.wave ([pol.dig[track] FROM 0 FOR snapshot.len],
                   xorig, yorig, xlen, ylen, subplot.num, data.out)
      yorig := (yorig - ylen)
    }}}
   SEQ track = 0 FOR num.tracks
    {{{  output gxo out
    SEQ
      plot.d.wave ([gxo.dig[track] FROM 0 FOR snapshot.len],
                   xorig, yorig, xlen, ylen, subplot.num, data.out)
      yorig := (yorig - 1.5(REAL32))
    }}}
:
}}}F
}}}
{{{  SC draw base
{{{F draw base
{{{  Header
```

-- Prints Error Results onto Graph.

```
}}}
{{{  PROC header
PROC draw.base (VAL INT num.tracks, INT subplot.num, mesg.num,
                VAL INT pulse.sep,
                VAL REAL32 gate.threshold, pol.threshold,
                VAL REAL32 gate.noise.pp, pol.noise.pp, displacement,
                VAL BOOL staggered,
                VAL INT sim.data.rate, max.bad, min.good,
                VAL []BYTE comment.text,
                VAL INT comment.len, VAL []REAL32 track.rate,
                VAL REAL32 rate, CHAN OF ANY data.out)
}}}
 #USE userio :
 {{{  PROC decl's
 {{{  SC text.new.line
 {{{F text.new.line
 {{{  Header
 -- Internal to draw.base
 }}}
 PROC text.new.line (REAL32 ycord)
   SEQ
     ycord := ycord - 0.5(REAL32)
 :
 }}}F
 }}}
 {{{  SC text.new.column
 {{{F text.new.column
 {{{  Header
 -- Internal to draw.base
 }}}
 PROC text.new.column (REAL32 xcord, ycord)
   SEQ
     xcord := xcord + 4.5(REAL32)
     ycord := 2.5(REAL32)
 :
 }}}F
 }}}
 {{{  SC write.mesg.int
 {{{F write.mesg.int
 {{{  Header
 -- Write Message with Integer.
 -- Internal to draw.base. Increment Message number.
 }}}
 PROC write.mesg.int (INT mesg.num,
                      VAL []BYTE string,
                      VAL INT value,
                      VAL REAL32 xcord, ycord,
                      CHAN OF ANY data.out)
   #USE userio :
   SEQ
     write.full.string (data.out, "MSG ")
     write.int (data.out, mesg.num, 0)
```

```
    write.full.string (data.out, " *"")
    write.full.string (data.out, string)
    IF
      value <> (-1) -- allows for no number being printed
        write.int (data.out, value, 0)
      TRUE
        SKIP
    write.full.string (data.out, "*", X=")
    write.real32 (data.out, xcord, 0, 0)
    write.full.string (data.out, ", Y=")
    write.real32 (data.out, ycord, 0, 0)
    write.text.line (data.out, ".")
    mesg.num := (mesg.num PLUS 1)
:
}}}F
}}}
{{{ SC write.mesg.float
{{{F write.mesg.float
{{{ Header
-- Write Message with Floating point number
-- Internal to draw.base. Increment Message number.
}}}
PROC write.mesg.float (INT mesg.num, VAL []BYTE string, VAL REAL32 value,
                       VAL REAL32 xcord, ycord, CHAN OF ANY data.out)
  #USE userio :
  SEQ
    write.full.string (data.out, "MSG ")
    write.int (data.out, mesg.num, 0)
    write.full.string (data.out, " *"")
    write.full.string (data.out, string)
    write.real32 (data.out, value, 1, 5)
    write.full.string (data.out, "*", X=")
    write.real32 (data.out, xcord, 0, 0)
    write.full.string (data.out, ", Y=")
    write.real32 (data.out, ycord, 0, 0)
    write.text.line (data.out, ".")
    mesg.num := (mesg.num PLUS 1)
:
}}}F
}}}
{{{ SC write track id
{{{F write track id
{{{ Header
-- Annotate Graphs according to Track Number.
-- Internal to draw.base
}}}
PROC write.track.id (INT mesg.num, VAL INT track,
                     VAL REAL32 xorig, yorig,
                     CHAN OF ANY data.out)
  #USE userio :
  SEQ
    write.full.string (data.out, "MSG ")
    write.int (data.out, mesg.num, 0)
    write.full.string (data.out, " *"T")
    write.int (data.out, (track PLUS 1), 0)
    write.full.string (data.out, "*", X=")
    write.real32 (data.out, xorig, 0, 0)
    write.full.string (data.out, ", Y=")
    write.real32 (data.out, yorig, 0, 0)
    write.text.line (data.out, ".")
    mesg.num := (mesg.num PLUS 1)
:
}}}F
}}}
}}}
REAL32 xtext, ytext :
SEQ
  {{{ continue
  write.full.string (data.out, "CONTINUE ")
  write.int (data.out, subplot.num, 0)
  write.text.line (data.out, ".")
  subplot.num := (subplot.num PLUS 1)
```

```
}}}
write.text.line (data.out, "GENERATE A PLOT.")
{{{  write general data
xtext, ytext := 2.0(REAL32), 2.5(REAL32)
{{{  first column
write.mesg.int (mesg.num, "SIM DATA RATE ", sim.data.rate, xtext, ytext, data.out)
text.new.line (ytext)
write.mesg.float (mesg.num, "GATE THRESH ", gate.threshold, xtext, ytext, data.out)
text.new.line (ytext)
write.mesg.float (mesg.num, "POL THRESH ", pol.threshold, xtext, ytext, data.out)
text.new.line (ytext)
write.mesg.float (mesg.num, "GATE NOISE PP ", gate.noise.pp, xtext, ytext, data.out)
text.new.line (ytext)
write.mesg.float (mesg.num, "POL NOISE PP ", pol.noise.pp, xtext, ytext, data.out)
text.new.line (ytext)
}}}
{{{  second column
write.mesg.float (mesg.num, "TRACK DISP ", displacement, xtext, ytext, data.out)
text.new.column (xtext, ytext)
{{{  staggered
IF
  staggered
    write.mesg.int (mesg.num, "DATA STAGGERED ", -1, xtext, ytext, data.out)
  TRUE
    write.mesg.int (mesg.num, "DATA NOT STAGGERED ", -1, xtext, ytext, data.out)
}}}
text.new.line (ytext)
write.mesg.int (mesg.num, "MAX BAD ", max.bad, xtext, ytext, data.out)
text.new.line (ytext)
write.mesg.int (mesg.num, "MIN GOOD ", min.good, xtext, ytext, data.out)
text.new.line (ytext)
{{{  comment
IF
  (comment.len MINUS 1) > 0
    write.mesg.int (mesg.num, [comment.text FROM 0 FOR (comment.len MINUS 1)],
                    -1, xtext, ytext, data.out)
  TRUE
    write.mesg.int (mesg.num, "-", -1, xtext, ytext, data.out)
}}}
text.new.column (xtext, ytext)
}}}
{{{  third column
SEQ track = 0 FOR num.tracks
  SEQ
    write.full.string (data.out, "MSG ")
    write.int (data.out, mesg.num, 0)
    write.full.string (data.out, " *"RATE T")
    write.int (data.out, (track PLUS 1), 0)
    write.full.string (data.out, " ")
    write.real32 (data.out, track.rate[track], 1, 5)
    write.full.string (data.out, "*", X=")
    write.real32 (data.out, xtext, 0, 0)
    write.full.string (data.out, ", Y=")
    write.real32 (data.out, ytext, 0, 0)
    write.text.line (data.out, ".")
    mesg.num := (mesg.num PLUS 1)
    text.new.line (ytext)
write.mesg.float (mesg.num, "OVERALL RATE ", rate, xtext, ytext, data.out)
text.new.column (xtext, ytext)
}}}
}}}
{{{  write track id's
xtext, ytext := 19.5(REAL32), 28.5(REAL32)
SEQ track = 0 FOR num.tracks
  {{{  output gate anotation
  SEQ
    write.track.id (mesg.num, track, xtext, ytext, data.out)
    ytext := (ytext - 2.5(REAL32))
  }}}
SEQ track = 0 FOR num.tracks
  {{{  output polarity anotation
  SEQ
```

```
            write.track.id (mesg.num, track, xtext, ytext, data.out)
            ytext := (ytext - 2.5(REAL32))
          }}}
          ytext := (ytext + 1.0(REAL32))
          SEQ track = 0 FOR num.tracks
            {{{  output gxo out
            SEQ
              write.track.id (mesg.num, track, xtext, ytext, data.out)
              ytext := (ytext - 1.5(REAL32))
            }}}
          }}}
          {{{  subplot
          write.full.string (data.out, "SUBPLOT ")
          write.int (data.out, subplot.num, 0)
          write.text.line (data.out, ".")
          }}}
        :
        }}}F
        }}}
        PAR
          {{{  gen text
          SEQ
            {{{  init
            xorig := 2.0(REAL32)
            yorig := 26.5(REAL32)
            subplot.num := 1
            mesg.num := 1
            {{{  send message to log
            write.full.string (to.log, "Waveforms filed in ")
            write.text.line (to.log, [waveform.filename FROM 0 FOR waveform.name.len])
            }}}
            }}}
            {{{  generate text
            SEQ
              write.text.line (to.waveforms, "ERROR REPORTING LEVEL 2.")
              write.text.line (to.waveforms, "PAGE LAYOUT HRV.")
              write.text.line (to.waveforms, "GENERATE A PLOT.")
              {{{  subplot
              write.full.string (to.waveforms, "SUBPLOT ")
              write.int (to.waveforms, subplot.num, 0)
              write.text.line (to.waveforms, ".")
              }}}
              {{{  draw graphs
              draw.graphs (pre.headamp, post.headamp, gate.ana, pol.ana,
                           gate.dig, pol.dig, gxo.dig, snapshot.len,
                           xorig, yorig, subplot.num, to.waveforms)
              }}}
              {{{  draw.base
              draw.base (num.tracks, subplot.num, mesg.num, pulse.sep,
                         gate.threshold, pol.threshold,
                         gate.noise.pp, pol.noise.pp, displacement,
                         staggered,
                         sim.data.rate, max.bad, min.good,
                         comment.text, comment.len,
                         track.rate, rate, to.waveforms)
              }}}
              write.endstream (to.waveforms)
            }}}
          }}}
          {{{  file text
          scrstream.to.server (to.waveforms, from.filer, to.filer, waveform.name.len,
                               waveform.filename, waveform.filer.result)
          }}}
        }}}
      }}}
      {{{  message
      print.elapsed.time (to.log, start.time)
      write.text.line (to.log, "        FINISHED ! ")
      }}}
    write.endstream (to.log)
  }}}
  {{{  send text to "LOG", copied to screen
```

```
          CHAN OF ANY to.log.fold :
          PAR
            SEQ
              scrstream.fan.out (to.log, to.log.fold, screen)
              write.endstream (to.log.fold)
            scrstream.to.file (to.log.fold, from.user.filer[1], to.user.filer[1],
                               "log", log.fold.num, log.fold.result)
        :::
      :::
    {{{ clean up
    keystream.sink (from.param.fold)
    :::
  :::
  {{{          using parameters read from fold
  keystream.from.file (from.user.filer[0], to.user.filer[0], from.param.fold,
                       top.fold, param.fold.result)
  :::
{{{ clean finish
write.full.string (screen, "Press any key.... ")
read.char (keyboard, char)
:::
```

-- End of Simulation Code.

```
{{{ SC antidisp
{{{F antidisp
{{{ Header
```

-- Displacement Compensation Scheme.

```
-- Uses track width, seperation and displacement.
-- Record pre- and post-compensation waveforms (in snapshot)
:::
#USE pseudort :
VAL num.tracks IS 2 :
PROC antidisp ([num.tracks]CHAN OF INT.OR.FLOAT data.in,
               VAL REAL32 w, TS, disp,
               [num.tracks][]REAL32 pre.snapshot, post.snapshot,
               VAL INT settle.time, snapshot.step.size,
               [num.tracks][9000]REAL32 uncomp.signal, comp.signal,
               [num.tracks]CHAN OF INT.OR.FLOAT data.out)
  {{{ decl's
  #USE t4math :
  VAL INT Track1 IS 0 :
  VAL INT Track2 IS 1 :
  VAL end IS ((SIZE pre.snapshot[0]) MINUS 1) :
  VAL track2.gain IS 1.2(REAL32) :
  [num.tracks]REAL32 data :
  [num.tracks]INT char :
  REAL32 on.track, next.track, coeff1, coeff2, adj.data.Track1, adj.data.Track2 :
  INT ptr, out.ptr, count, snap.count, num.terminated :
  BOOL settling, capturing, running :
  :::
  SEQ
    {{{ init
    {{{ calc coeff's
    on.track := (w - disp)
    next.track := ((w + disp) - TS)
    IF
      next.track < zero
        next.track := zero
      TRUE
        SKIP
    coeff1 := (w / on.track)
    coeff2 := (next.track / on.track)
    :::
    SEQ track = 0 FOR num.tracks
      char[track] := 0
    count := 0
    ptr, out.ptr := 0, 0
```

```
snap.count := 0
num.terminated := 0
settling := TRUE
capturing := FALSE
running := TRUE
}}}
WHILE running
  SEQ
    {{{  get data
    PAR track = 0 FOR num.tracks
      data.in[track] ? CASE
        int ; char[track]
          SKIP
        float ; data[track]
          SKIP
    }}}
    {{{  check for 'terminated'
    SEQ track = 0 FOR num.tracks
      IF
        char[track] = 0 -- initial value
          SKIP
        char[track] = terminate
          num.terminated := (num.terminated PLUS 1)
        TRUE
          SEQ
            data.out[track] ! int ; char[track]
            char[track] := 0 -- back to initial value again
    }}}
    IF
      num.terminated = 0
        {{{  process data
        SEQ
          {{{  de-mix signals
          data[Track2] := (data[Track2] * track2.gain)
          {{{  record input data
          uncomp.signal[Track1][out.ptr] := data[Track1]
          uncomp.signal[Track2][out.ptr] := data[Track2]
          }}}
          adj.data.Track1 := (coeff1 * data[Track1])
          adj.data.Track2 := (coeff1 * (data[Track2] + (coeff2 * data[Track1])))
          {{{  record compensated data
          comp.signal[Track1][out.ptr] := adj.data.Track1
          comp.signal[Track2][out.ptr] := adj.data.Track2
          out.ptr := (out.ptr PLUS 1)
          }}}
          }}}
          {{{  output signals
          PAR
            data.out[Track1] ! float ; adj.data.Track1
            data.out[Track2] ! float ; adj.data.Track2
          }}}
          {{{  snap section of the number stream
          IF
            settling
              {{{
              SEQ
                count := count PLUS 1
                IF
                  settle.time > count
                    SKIP
                  TRUE
                    SEQ
                      settling := FALSE
                      capturing := TRUE
              }}}
            capturing
              {{{
              SEQ
                snap.count := (snap.count PLUS 1)
                IF
                  snap.count = snapshot.step.size
                    SEQ
```

```
                    pre.snapshot[Track1][ptr] := data[Track1]
                    post.snapshot[Track1][ptr] := adj.data.Track1
                    pre.snapshot[Track2][ptr] := data[Track2]
                    post.snapshot[Track2][ptr] := adj.data.Track2
                    {{{ increment ptr
                    IF
                      ptr <> end
                        ptr := ptr PLUS 1
                      TRUE
                        capturing := FALSE
                    }}}
                    snap.count := 0
                  TRUE
                    SKIP
              }}}
            TRUE
              SKIP
          }}}
        }}}
      TRUE
        {{{ sink rest of data until all terminated
        SEQ
          WHILE num.terminated <> num.tracks
            ALT track = 0 FOR num.tracks
              data.in[track] ? CASE
                int ; char[track]
                  {{{
                  IF
                    char[track] = terminate
                      num.terminated := (num.terminated PLUS 1)
                    TRUE
                      SKIP
                  }}}
                float ; data[track]
                  SKIP
          running := FALSE
        }}}
    {{{ pass on 'terminate'
    PAR track = 0 FOR num.tracks
      data.out[track] ! int ; terminate
    }}}
:
}}}F
}}}
```